COO-1018-1194

ILLIAC III COMPUTER SYSTEM MANUAL: ARITHMETIC UNITS

VOLUME I

by

Daniel E. Atkins

December 1969

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

COO-1018-1194

Report No. 366

ILLIAC III COMPUTER SYSTEM MANUAL: ARITHMETIC UNITS*

VOLUME I

by

Daniel E. Atkins

December 1969

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

*This work was supported by the U.S. Atomic Energy Commission under Contract No. USAEC AT(11-1)-1018.

ACKNOWLEDGMENT

The conceptual design of the Arithmetic Units is the work of James E. Robertson and the author. Mrs. Tuh-Kai Koo wrote extensive simulation programs which were used in validating the arithmetic algorithms. Bruce H. McCormick provided conceptual guidance and contributed greatly in the editing of this volume.

The typing of this report is the work of Mrs. Mariam Coleman and Mrs. Donna J. Stutz. Illustrations were prepared by John Otten, Eric Storm, Ron Morrison, Stan Zundo, Bill Giblin, and Mary Ann Hagen assisted in proofreading.

Larry Byers assisted in the preparation of detailed logic drawings; Bob Thomson assisted by John Fileccia, bore the weight of the arduous task of layout and wiring table generation. The "POOL" programs developed under the direction of S. Paul Krabbe were very valuable in the preparation of wiring tables.

INTRODUCTION

This report comprises the first volume of the manual for the Illiac III Arithmetic Units. It corresponds to the introduction (Section 1) and Internal Static Description (Section 2) of the overall outline.

CUTLINE OF VOLUME I

2.02 REGISTERS
- 2.2.1 M-REGISTER
  - 2.2.1.1 GENERAL
  - 2.2.1.2 IMPLEMENTATION
  - 2.2.1.3 PL/1 DESCRIPTION OF SIGNAL NAMES
- 2.2.2 LS-REGISTER
  - 2.2.2.1 GENERAL
  - 2.2.2.2 IMPLEMENTATION
  - 2.2.2.3 PL/1 DESCRIPTION OF SIGNAL NAMES
- 2.2.3 UM-REGISTER
  - 2.2.3.1 GENERAL
  - 2.2.3.2 IMPLEMENTATION
  - 2.2.3.3 PL/1 DESCRIPTION OF SIGNAL NAMES
- 2.2.4 LS-REGISTER
  - 2.2.4.1 GENERAL
  - 2.2.4.2 IMPLEMENTATION
  - 2.2.4.3 PL/1 DESCRIPTION OF SIGNAL NAMES
- 2.2.5 LM-REGISTER
  - 2.2.5.1 GENERAL
  - 2.2.5.2 IMPLEMENTATION
  - 2.2.5.3 PL/1 DESCRIPTION OF SIGNAL NAMES
- 2.2.6 LH-REGISTER
  - 2.2.6.1 GENERAL
  - 2.2.6.2 IMPLEMENTATION
  - 2.2.6.3 PL/1 DESCRIPTION OF SIGNAL NAMES
- 2.2.7 LC-REGISTER
  - 2.2.7.1 GENERAL
  - 2.2.7.2 IMPLEMENTATION
  - 2.2.7.3 PL/1 DESCRIPTION OF SIGNAL NAMES
- 2.2.8 LM-REGISTER
  - 2.2.8.1 GENERAL
  - 2.2.8.2 IMPLEMENTATION
  - 2.2.8.3 PL/1 DESCRIPTION OF SIGNAL NAMES
- 2.2.9 LC-REGISTER
  - 2.2.9.1 GENERAL
  - 2.2.9.2 IMPLEMENTATION
  - 2.2.9.3 PL/1 DESCRIPTION OF SIGNAL NAMES

2.03 V-BUS INTERFACE
- 2.3.1 GENERAL
- 2.3.2 INPUT TO THE ARITHMETIC UNITS VIA THE V-BUS
- 2.3.3 IMPLEMENTATION
- 2.3.4 PL/1 DESCRIPTION OF SIGNAL NAMES

2.04 XT-BUS INTERFACE
- 2.4.1 GENERAL
- 2.4.2 OUTPUT FROM THE ARITHMETIC UNITS VIA THE XT-BUS
- 2.4.3 IMPLEMENTATION
- 2.4.4 PL/1 DESCRIPTION OF SIGNAL NAMES

2.05 SIGNED-DIGIT SUBTRACTERS
- 2.5.1 GENERAL
- 2.5.2 IMPLEMENTATION

2.06 (UNASSIGNED)

LIST OF FIGURES

LIST OF TABLES

INDEX TO LOGIC DRAWINGS

****** AS OF 10 MARCH 1969 THIS VERSION OF THE INDEX REFLECTS THE STATU
OF THE DRAWINGS FROM WHICH THE FINAL WIRING TABLES FOR THE
PROCESSING HARDWARE (NOT CONTROL) WERE GENERATED. ******

THE ORIGINALS OF THESE DRAWINGS, EXCEPT AS NOTED, ARE ON FILE IN THE 101
DRAFTING SECTION (ROOM 50 OCL)

SPECIFICATION OF CARD FORMAT

CARD COL. NO.	CONTENTS
01-06	DRAWING NUMBER
07	BLANK OR '*' IF ORIGINAL OF DRAWING IS IN WORKING FILE IN RATHER THAN IN DRAFTING.
08-09	THE INITIALS AU
10	BLANK
11-67	DESCRIPTION OF DRAWING
68-72	THE DATE OF THE MOST RECENT VERSION OF THE DRAWING. (NO. OF MONTH, NO. OF DAY, LAST DIGIT OF YEAR)
73	BLANK
74	1 IF PRELIMINARY DRAWING IS COMPLETE, BLANK OTHERWISE.
75	1 IF PACKAGING OF THE LOGIC ON THE DRAWING HAS BEEN SPECIFIED, BLANK OTHERWISE.
76	1 IF WIRING TABLE HAS BEEN GENERATED FOR DRAWING, BLANK OTHERWISE.
77-80	NOT DEFINED

NOTE 'DATE' IS THE DATE OF THE MOST RECENT VERSION OF THE DRAWING.

ATKINS DWNG. NO.	DESCRIPTION	DATE
210-01 AU	INPUT GATES TO MODEL DIVISION	0307
210-02 AU	MODEL DIVISION DIVISOR INTERVAL SELECT AND ASSIMILATION	0307
210-03 AU	QUOTIENT SELECT DIODE MATRIX	0307
210-04 AU	MODEL DIVISION QUOTIENT BUFFER	0307
210-05 AU	MODEL DIVISION OUTPUT GATING	0307
210-06 AU	MODEL DIVISION QUOTIENT BUFFER INPUT TO UQ AND UH	0307
211-01 AU	UQ AND UH ZERO DETECT	0307
211-02 AU	FLAG REGISTERS (FA AND FB) AND FLAG MATCH LOGIC	0307
211-03 AU	UQ REGISTER ALL ONES DETECT	0307
211-04*AU	PARITY ERROR FLIP-FLOP	1015
211-05*AU	IV AND NT REGISTERS WITH NT DECODER	0213
211-06*AU	INSTRUCTION VARIANT DECODER	0212
212-01*AU	ILLIAC III EXPONENT ARITHMETIC UNIT - BLOCK DIAGRAM	1127
212-02*AU	EUM AND EUQ REGISTERS WITH IN/OUT GATING	1127
212-03*AU	EXPONENT UNIT ADDER	1127
212-04*AU	EUL REGISTER-COUNTER	0924
212-05*AU	EAU CONDITION DETECT	1127
213-01 AU	UNASSIGNED	
213-02*AU	CONTROL POINT VIN-01, VIN-02	1003

03*AU	CONTROL POINT VIN-03	10098	1
04*AU	CONTROL POINT VIN-04, VIN-05	10098	1
01 AU	ARITHMETIC UNIT CARD LAYOUT - TOP BAY	07188	1
02 AU	ARITHMETIC UNIT CARD LAYOUT - BOTTOM BAY	07188	1
01 AU	M REGISTER, POSITIONS 09-16	03079	111
02 AU	M REGISTER, POSITIONS 17-32	03079	111
03 AU	M REGISTER, POSITIONS 33-48	03079	111
04 AU	M REGISTER, POSITIONS 49-64	03079	111
05 AU	M REGISTER SELECTOR AND GATE DRIVERS	03079	111
06 AU	M REGISTER OUTPUT INVERTERS	03079	111
07*AU	M SELECTOR FLIP/FLOPS	12178	1
01 AU	US REGISTER WITH SELECTORS, POSITIONS 01-16	03079	111
02 AU	US REGISTER WITH SELECTORS, POSITIONS 17-32	03079	111
03 AU	US REGISTER WITH SELECTORS, POSITIONS 33-48	03079	111
04 AU	US REGISTER WITH SELECTORS, POSITIONS 49-64	03079	111
05 AU	US LOAD AND SELECTOR DRIVERS (SAME AS 223-05)	03079	111
06 AU	US REGISTER OUTPUT GATING, POSITIONS 01-18	03079	111
07 AU	US REGISTER OUTPUT GATING, POSITIONS 19-36	03079	111
08 AU	US REGISTER OUTPUT GATING, POSITIONS 37-54	03079	111
09 AU	US REGISTER OUTPUT GATING, POSITIONS 55-64	03079	111
10 AU	LS, UM REGISTER OUTPUT GATING DRIVERS (SAME AS 223-06)	03079	111
01 AU	UM REGISTER WITH SELECTORS, POSITIONS 01-16	03079	111
02 AU	UM REGISTER WITH SELECTORS, POSITIONS 17-32	03079	111
03 AU	UM REGISTER WITH SELECTORS, POSITIONS 33-48	03079	111
04 AU	UM REGISTER WITH SELECTORS, POSITIONS 49-64	03079	111
05 AU	UM LOAD AND SELECTOR DRIVERS (SAME AS 222-05)	03079	111
06 AU	UM REGISTER OUTPUT GATING, POSITIONS 01-32	03079	111
07 AU	UM REGISTER OUTPUT GATING, POSITIONS 33-64	03079	111
08 AU	LS, UM REGISTER OUTPUT GATING DRIVERS (SAME AS 222-10)	03079	111
01 AU	LS REGISTER, POSITIONS 01-08 (DETAIL OF CARD)	03079	111
02 AU	LS REGISTER, POSITIONS 09-24	03079	111
03 AU	LS REGISTER, POSITIONS 25-40	03079	111
04 AU	LS REGISTER, POSITIONS 41-56	03079	111
05 AU	LS REGISTER, POSITIONS 57-64	03079	111
06 AU	LM, LS REGISTER GATE DRIVERS (SAME AS 225-06)	03079	111
01 AU	LM REGISTER, POSITIONS 01-08 (DETAIL OF CARD)	03079	111
02 AU	LM REGISTER, POSITIONS 09-24	03079	111
03 AU	LM REGISTER, POSITIONS 25-40	03079	111
04 AU	LM REGISTER, POSITIONS 41-56	03079	111
05 AU	LM REGISTER, POSITIONS 57-64	03079	111
06 AU	LM, LS REGISTER GATE DRIVERS (SAME AS 224-06)	03079	111
01 AU	UH REGISTER WITH SELECTORS, POSITIONS 01-16	03079	111
02 AU	UH REGISTER WITH SELECTORS, POSITIONS 17-32	01048	11
03 AU	UH REGISTER WITH SELECTORS, POSITIONS 33-48	03079	111
04 AU	UH REGISTER WITH SELECTORS, POSITIONS 49-64	03079	111
05 AU	UH LOAD AND SELECTOR DRIVERS	03079	111
06 AU	LMDLH7 OF LH SELECTOR	03079	111
07 AU	LHLIUH OF UH SELECTOR	03079	111
01 AU	UQ REGISTER WITH SELECTORS, POSITIONS 01-16	03079	111
02 AU	UQ REGISTER WITH SELECTORS, POSITIONS 17-32	03079	111
03 AU	UQ REGISTER WITH SELECTORS, POSITIONS 33-48	03079	111
04 AU	UQ REGISTER WITH SELECTORS, POSITIONS 49-64	03079	111
05 AU	UQ LOAD AND SELECTOR DRIVERS	03079	111
06 AU	LQLIUQ, LQRIUQ, AND SETUQCNE OF UQ SELECTOR	03079	111
07*AU	UQ SELECTOR FLIP/FLOPS	12178	1

228-01 AU LH REGISTER, POSITIONS 01-08 (DETAIL OF CARD)	03079
228-02 AU LH REGISTER, POSITIONS 09-24	03079
228-03 AU LH REGISTER, POSITIONS 25-40	03079
228-04 AU LH REGISTER, POSITIONS 41-56	03079
228-05 AU LH REGISTER, POSITIONS 57-64	03079
228-06 AU LH LOAD DRIVERS (SAME AS 229-06)	03079
229-01 AU LQ REGISTER, POSITIONS 01-08 (DETAIL OF CARD)	03079
229-02 AU LQ REGISTER, POSITIONS 09-24	03079
229-03 AU LQ REGISTER, POSITIONS 25-40	03079
229-04 AU LQ REGISTER, POSITIONS 41-56	03079
229-05 AU LQ REGISTER, POSITIONS 57-64	03079
229-06 AU LQ LOAD DRIVERS (SAME AS 228-06)	03079
230-01 AU BLK DIAGRAM OF V-BUS AND X-BUS	03079
230-02 AU V-BUS TERMINATORS, X-BUS DRIVERS, BYTES 1,2,3,4	03079
230-03 AU V-BUS PARITY CHECKING, BYTES 1,2	03079
230-04 AU V-BUS PARITY CHECKING, BYTES 3,4	03079
230-05 AU V-BUS TERMINATORS, X-BUS DRIVERS, BYTE 0	12297
240-01 AU XT BUS SELECTOR	03079
240-02 AU XT BUS PARITY GENERATOR, BYTES 1,2	03079
240-03 AU XT BUS PARITY GENERATOR, BYTES 3,4	03079
240-04 AU GV AND GXT SELECTOR AND GATE DRIVERS	03079
250-01 AU SIGNED DIGIT SUBTRACTOR S1, 1ST 4 POSITIONS OF BYTES	03079
250-02 AU SIGNED DIGIT SUBTRACTOR S1, 2ND 4 POSITIONS OF BYTES	03079
250-03 AU SIGNED DIGIT SUBTRACTOR S2, 1ST 4 POSITIONS OF BYTES	03079
250-04 AU SIGNED DIGIT SUBTRACTOR S2, 2ND 4 POSITIONS OF BYTES	03079
250-05 AU SIGNED DIGIT SUBTRACTOR S3, 1ST 4 POSITIONS OF BYTES	03079
250-06 AU SIGNED DIGIT SUBTRACTOR S3, 2ND 4 POSITIONS OF BYTES	03079
250-07 AU SIGNED DIGIT SUBTRACTOR S4, 1ST 4 POSITIONS OF BYTES	03079
250-08 AU SIGNED DIGIT SUBTRACTOR S4, 2ND 4 POSITIONS OF BYTES	03079
250-09 AU DRIVERS FOR SDS CONTROL AND M SHIFT ARRAY (S1 AND S2)	03079
250-09 AU S1,S2,S3 NEG AND G DRIVERS OBSOLETE	02158
250-10 AU DRIVERS FOR SDS CONTROL AND M SHIFT ARRAY (S3 AND S4)	03079
250-10 AU S3,S4 NEG AND G DRIVERS OBSOLETE	02158
250-11 AU OV CORRECTION LOGIC, S1, S2, S3, AND S4	03079
NOTE THERE IS NO 26C- SERIES.	
270-01 AU PROPAGATION LOGIC, FIRST LEVEL POSITIONS 01-12	03079
270-02 AU PROPAGATION LOGIC, FIRST LEVEL POSITIONS 13-24	03079
270-03 AU PROPAGATION LOGIC, FIRST LEVEL POSITIONS 25-36	03079
270-04 AU PROPAGATION LOGIC, FIRST LEVEL POSITIONS 37-48	03079
270-05 AU PROPAGATION LOGIC, FIRST LEVEL POSITIONS 49-60	03079
270-06 AU PROP. LOGIC,1ST LEVEL POS 61-64,2ND LEVEL POS 01-32	03079
270-07 AU PROP. LOGIC,2ND LEVEL POS 33-64,3RD LEVEL POS 01-64	03079
280-01 AU M SHIFT ARRAY POSITIONS Y(1,1) - Y(1,8)	03079
280-02 AU M SHIFT ARRAY POSITIONS Y(1,9) - Y(1,40)	03079
280-03 AU M SHIFT ARRAY POSITIONS Y(1,41)- Y(1,64)	03079
280-04 AU M SHIFT ARRAY POSITIONS Y(2,1) - Y(2,32)	03079
280-05 AU M SHIFT ARRAY POSITIONS Y(2,33)- Y(2,64)	03079
280-06 AU M SHIFT ARRAY POSITIONS Y(3,1) - Y(3,32)	03079
280-07 AU M SHIFT ARRAY POSITIONS Y(3,33)- Y(3,64)	03079
280-08 AU M SHIFT ARRAY POSITIONS Y(4,1) - Y(4,32)	03079
280-09 AU M SHIFT ARRAY POSITIONS Y(4,33)- Y(4,64)	03079
280-10 AU M SHIFT DRIVERS REPLACED BY 250-09, 250-10	03268
280-11 AU M SHIFT CRCYL GATES	03079
290-01 AU MULTIPLIER RECODE	03079
290-02 AU MULTIPLY EXTENDED PRECISION LOGIC	03079

```
**** NOTE  A DRAWING NUMBER PREFIXED WITH 'FC' DENOTES A CONTROL FLOW CHART *
-01 *AU V-BUS INPUT CONTROL SEQUENCE  VIN-01,-02,-03                  10028 1
-02 *AU V-BUS INPUT CONTROL SEQUENCE  VIN-04,-05                      10068 1
-03 *AU ADD,SUBTRACT, OR COMPARE (ASC)                                12208 1
-04 *AU ALIGN UF, ALIGN UG, UH TO UG                                  03209 1
-05 *AU CALCULATION CONTROL FOR ASC (CAL)                             03209 1
-06 *AU NORM UG                                                       03209 1
-07 *AU SET FLAGS                                                     03209 1
-08 *AU MULTIPLY (MPY)                                                03149 1
-09 *AU MULTIPLY END (MPYEND)                                         03209 1
                        ****** END ******
```

1.0 Introduction

1.1 Purpose, Scope and Organization of the Manual

This manual is a working document for the design, construction, checkout, and maintenance of the arithmetic units for Illiac III. It is an evolving document and will be frequently updated. Readers are encouraged to notify the author of errors or suggest clarifications.

Although the manual is intended to be essentially complete in itself there are supporting documents available to readers with special interests. These are described below:

Detailed Logic Drawings - Typical cross sections of logic are shown in the manual; however, due to their bulk, the detailed logic drawings are not included. The originals of these drawings (see Drawing Index in Section 0.0) are maintained by the Illiac III drafting section and will generally be available only to those directly involved in construction, checkout, or maintenance.

Engineering Manual - The Engineering Manual contains construction details for all of the printed circuit boards with which Illiac III is constructed. Copies of the logic diagrams for relevant boards are included in the AU Manual. The Engineering Manual is also maintained by the Drafting Section and is available on a need-to-know basis.

DCS Report No. 333: "Design of the Arithmetic Units of Illiac III: Use of Redundancy and Higher-Radix Methods", by D. E. Atkins - In keeping with the experimental nature of Illiac III, the arithmetic units are intended to be a practical testing ground for recent theoretical work in computer arithmetic. This report is to represent the more theoretical aspects of the design, especially the use of a redundant number representation. The use of redundancy is a prime factor in obtaining high-speed operation. This report is readily available from the Department of Computer Science, Mailing Center, Room 222 DCL, University of Illinois, Urbana, Illinois 61801. The

AU Manual itself is primarily intended to be a design description as opposed to a design justification.

Publication: "Higher Radix Division Using Estimates of the Divisor and Partial Remainders", by D. E. Atkins - This paper presents the theoretical basis for the division scheme implemented in Illiac III. It is published in the IEEE Transactions on Computers, Vol. C-17, No. 10, (October 1968), pp. 925-934. Reprints are available from the author, or the Illiac III office, 297 Digital Computer Laboratory.

The arithmetic unit manual is divided into four major sections: an introduction, an internal static description, an operation description and a presentation of AU simulation programs.

The internal static description, Section 2, begins by presenting the blocks diagram of an arithmetic unit and then describes each sub-system of the unit. This description is presented at three levels of detail: first, as a verbal function description; secondly, as a block diagram of the sub-systems; and thirdly, as a detailed view of the actual logic. For most sub-systems only a typical section of the logic is shown.

Likewise the operational description, Section 3, is presented in steps of increasing detail. Each order executed in au AU is described as a sequence of sub-operations. These sub-operations are defined verbally and then with detailed flow charts and flow tables.

Section 4 is a alphabetized list of signal names together with the PL/1 definition.

1.2 Conventions

The logic symbols used in this manual conform to MIL-STD-806B, Graphic Symbols for Logic Diagrams. Numbers appearing within the symbol (if any) specify the Illiac III card type as defined in the Engineering Manual.

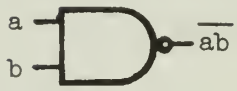
Positive logic is used throughout. The symbol, 1, denotes a logically true state and corresponds to a nominal +6 volts. The symbol, 0, denotes a logically false state and corresponds to a nominal 0 volts. Logic notation is defined below:

Function	Truth Table for Variables a, b	Logic Symbol
AND	a b | ab 0 0 | 0 0 1 | 0 1 1 | 1 1 0 | 0	
OR	a b | avb 0 0 | 0 0 1 | 1 1 1 | 1 1 0 | 1	
NEGATION or COMPLEMENTATION	a $\bar{a}$ 0 1 1 0	 

Function	Truth Table for Variables a,b	Logic Symbol

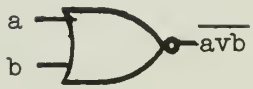
NAND
(Not AND)

a	b	$\overline{ab}$
0	0	1
0	1	1
1	0	1
1	1	0



NOR
(Not OR)

a	b	$\overline{avb}$
0	0	1
0	1	0
1	1	0
1	0	0



EXCLUSIVE OR

a	b	a⊕b
0	0	0
0	1	1
1	1	0
1	0	1

(no specific symbol)

EQUIVALENCE
(Complement of
EXCLUSIVE OR)

a	b	a≡b
0	0	1
0	1	0
1	1	1
1	0	0

(no specific symbol)

The arithmetic unit has been simulated using PL/1. It was found that this language is useful in describing both structure of registers, buses, etc. and the function of the various control signals. The description of each subsystem therefore includes a PL/1 description of signal names relevant to the structure. The following conventions have been adopted:

1. Registers and buses are defined as bit strings. Example:
   DECLARE US BIT (64);/* TRUE OUTPUTS OF US-REGISTER*/
   DECLARE USSEL BIT (64);/*TRUE OUTPUTS OF US-SELECTOR*/

2. Subsections of a register or bus (any bit string) are defined using the PL/1 function, SUBSTR. The general form of calling is SUBSTR (S, i, j) where S is the name of a bit string, i is the first position of the substring, and j is the length of the substring. Example:
   SUBSTR (US, 1, 32) is the left half of the US Register.

3. Signals which are common across many positions of a register, selector, etc. are divided into sub-signals. This is usually done only because of loading considerations but in some cases the sub-signals will be operated independent of each other. Sub-signals are designated in the form <signal name> nm, where nm denotes that the signal is common across byte n through byte m. If nm are not given, then the signal name is operative across all bytes, 0 through 7. Example:
   LDUS01 /*LOAD US-REGISTER BYTES 0 AND 1 */

4. The function of signal names are defined as procedures. The name of the procedure is the name of the signal. Example:
   LDUS01: /*USSEL BYTES 0-1 TO US BYTES 0-1 */ PROCEDURE;
   SUBSTR (US, 1, 16) = SUBSTR (USSEL, 1, 16); END;
   LDUS: PROCEDURE; CALL LDUS01; CALL LDUS23; CALL LDUS45;
   CALL LDUS67; END;

5. Shifting is accomplished by use of SUBSTR and concatenation (||).

```
LSL8US: PROCEDURE;/*LS LEFT 8 TO US */
US = SUBSTR (LS, 9, 56) || '00000000';
END;
```

6. The PL/1 logical operators are as follows:

|	IS OR
&	IS AND
¬	IS NEGATION

1.3 Brief Description of Interaction with Taxicrinic Processors

The two identical Arithmetic Units (AU) perform most of the arithmetic operations in the ILLIAC III system. Integer addition and subtraction, as well as several unary operations form exceptions and are actually executed in the Taxicrinic Processors. (These exceptions are noted in Table 1.5.2

The prime responsibility of the Arithmetic Units is the high-speed execution of floating point arithmetic operations. The units also provide facilities for integer multiplication and division and conversions from one number-type to another, e.g., floating to long fixed.

As in the case with all other units of the system, communication with the processors is via the Exchange Net. The Arithmetic Units interact primarily with the Taxicrinic Processors (TP), although paths are also available between the AU's and the I/O Processor. The Exchange Net assigns an AU to a requesting processor and also returns the results of an arithmetic operation to the processor which initiated the operation. Figure 1.3.1 illustrates the location of the AU's in the overall system.

The following is a general overview of the operation of the Arithmetic Units within the ILLIAC III System. The execution of an arithmetic instruction begins in a Taxicrinic Processor. The operands are assumed to be in the top of the operand stack of the TP. When a TP encounters an arithmetic instruction (an instruction with the mnemonic byte of the form 10XXXXXX1*it determines whether an AU is required to execute the operation. If this is the case, the TP places a request with the Exchange Net, which in turn locates and assigns an AU which is not in use. The TP then sends the AU a control byte containing the instruction variant (IV): add, subtract, etc. and the number type (NT): fixed, floating, etc, together with the operands.

---

*The last bit is a flag bit.

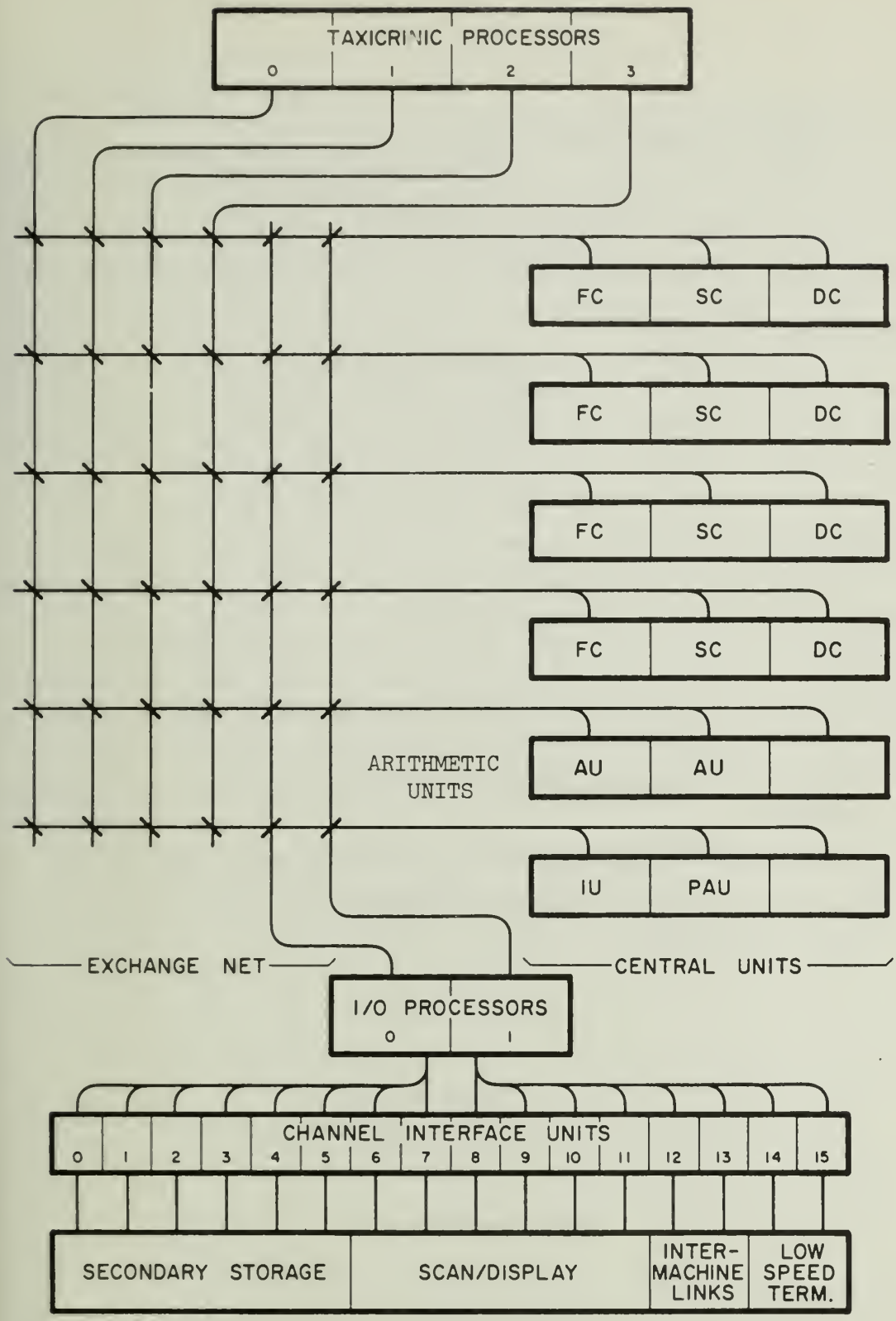


Figure 1.3.1 - Schematic of Illiac III Computer

Since the data paths through the Exchange Net are one word wide (4 bytes) and since in general, the operands consist of more than one full word, a series of transmissions is required. Upon rapid, initial decoding of the instruction variant and number type, the AU loads the operands as received, a word at a time, into the appropriate registers.

When all operands have been received, the TP-AU path is broken and the AU proceeds with execution of the operation. When the result has been formed, the AU notifies the Exchange Net, which in turn accesses the TP which initiated the AU operation. The result is then returned, a word at a time, to the TP. If an error condition such as OVERFLOW has occurred, a "1" appears on a designated line in the AU to TP control byte. The flags of the erroneous result being returned are set so as to indicate the nature of the error. Upon receiving the result, the TP places it in the top of the operand stack and continues to the next instruction. The AU is released and is available for the next assignment.

The expected execution time for floating point (56 bit mantissa) addition, subtraction, and multiplication is 3-6 μsec. and 8-9 μsec. for division.

1.4 Arithmetic Data Formats

There are four types of arithmetic data used in Illiac III. The attributes of these data are as follows:

Base = binary or decimal
Scale = fixed point or floating point
Mode = real
Precision = 16 bit integer, or
32 bit signed integer, or
56 bit signed fraction
and 7 bit chracteristic, or
14 decimal digit signed integer.

These attributes have been combined to form four different number types as described in the remainder of this section.

Although not illustrated in the previous figures, a flag bit is associated with each byte of the four number types described. Operands used in arithmetic operations may have flags set, and thus the question arises as to the flag setting of an arithmetic result. The flag bits of numbers produced by arithmetic operations will have the following significance.

a) For unary operations (other than number type conversions) the flags of the operand are unchanged.
b) For operations with two or more operands and no error conditions, the flag setting of the result is the flag setting of one of the operands.
c) For comparison instructions, the flags transmit the result of the comparison from the Arithmetic Unit (AU) to the Taxicrinic Processor (TP) via the Exchange Net (XN).
d) For operations resulting in error conditions, the flags transmit the type error condition from the AU to the TP via the XN.

The significance of arithmetic flags is described further in Section 1.5.10.

1.4.1 Short Fixed Point

Base = binary
Scale = fixed
Mode = real
Precision = 16 bit, unsigned integer if an address.
15 bit, signed integer otherwise.

A short fixed point number is a half-word binary integer which is treated as signed or unsigned depending upon its use. When used as an address, a short fixed point number will always be considered positive and since all 16 bits may be required to represent the magnitude, no sign bit is explicitly specified.

Addition of addresses, as performed in the pointer modification operation ADDITION, is computed Mod (2^{16}), in effect allow implicit address subtraction. Subtraction of addresses, as performed in the pointer modification operation CONDITIONAL SUBTRACTION, is performed as a two's complement subtraction, Mod 2^{16}. The results will always be treated as a positive integer.

Numbers of the short fixed point type may also be used for other than addresses in any of the 13 arithmetic operations. In this case, the most significant (MS) bit will be treated as a sign bit and a negative number will be represented in two's complement form with a sign bit of 1. The range of these integers is -2^{15} to $+(2^{15}-1)$ i.e., -32,768 to +32,767 and overflow will be checked.

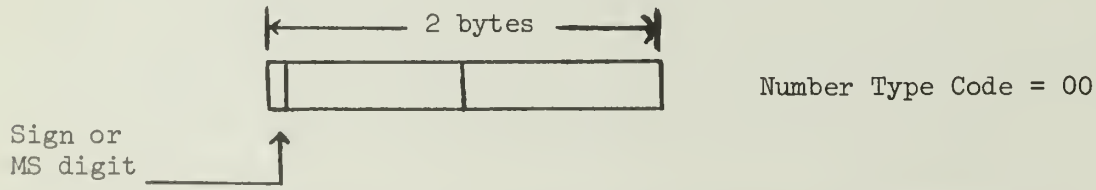


Figure 1.4.1.1 - Short Fixed Point Format

1.4.2 Long Fixed Point

Base = binary

Scale = fixed

Mode = real

Precision = 31 bit, signed integer

A long fixed point number is a full-word signed integer. Positive numbers are represented in true binary notation with a sign bit of zero. Negative numbers are represented in two's complement notation. The range of these integers is -2^{31} to $+(2^{31}-1)$, i.e., -2,147,483,648 to +2,147,483,647 and overflow will be checked.

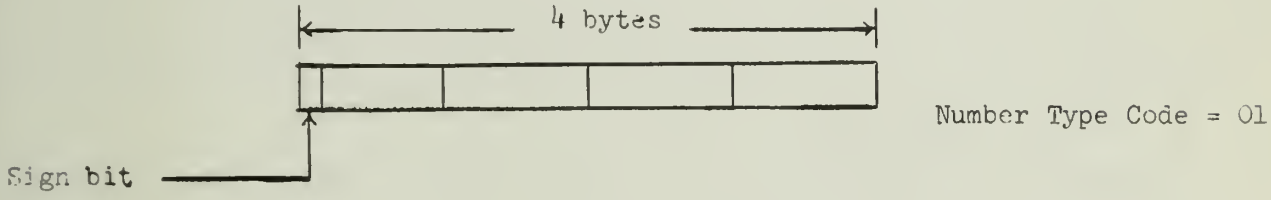


Figure 1.4.2.1 - Long Fixed Point Format

1.4.3 Floating Point

Base = binary

Scale = floating

Mode = real

Precision = 56 bit, signed fraction

7 bit characteristic

Floating point numbers are a double word in length, subdivided as indicated below:

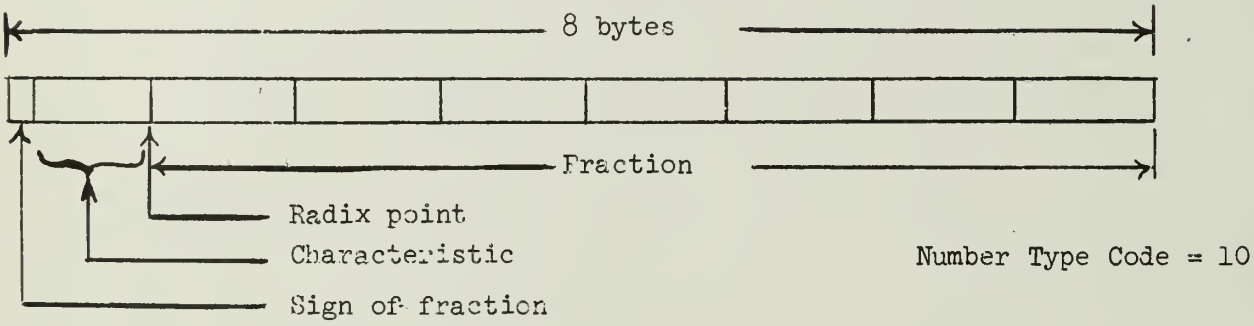


Figure 1.4.3.1 - Floating Point Format

The first bit is the sign of the fraction. The fraction is always in true representation, i.e., the fraction of negative numbers is carried in positive form. The fraction is expressed in base 16, hexadecimal form and therefore consists of 14 hexadecimal digits. In hexadecimal representation, the characteristic represents the power to which 16 must be raised to express the true magnitude of the number. The 7 bits of the characteristic are treated as an excess 64 number with range -64 to +63 corresponding to the binary values 0 through 127, respectively. The characteristic zero, for example, is represented as 1000000.

A floating point number in main store or produced as the result of a floating point arithmetic operation will always be normalized. For the hexadecimal representation, this means that the fraction will have a non-zero, high-order hexadecimal digit. This type normalization permits the three high order bits of a normalized number to be zero. Normalization is not programmable.

Under the normalization described above, the range covered by this notation is 16^{-65} to $(1 - 16^{-14}) \times 16^{63}$, which is approximately 5.4×10^{-79} to 7.2×10^{75}. The binary representation of these maximum and minimum values are shown in Table 1.4.3.1.

A floating point zero will be represented as a number with the most negative, zero fraction, and positive sign as indicated in the figure below. All zero results will be returned from the AU in this form.

DESCRIPTION	NUMBER	POWER OF 16	S	CHAR	FRACTION
Maximum positive number	$+7.2 \times 10^{75}$	$= (1-16^{-14}) \times 16^{63}$	0	1111111	All 1's
1.0	1.0×10^{0}	$= 1/16 \times 16^{1}$	0	1000001	00010...0
0.5	0.5×10^{0}	$= 1/2 \times 16^{0}$	0	1000000	10000...0
Minimum positive number	5.4×10^{-79}	$= 16^{-1} \times 16^{-64}$	0	0000000	000100...0
True zero	+0.0	$= 0 \times 16^{-64}$	0	0000000	All 0's

Table 1.4.3.1 - Examples of Floating Point Representation

1.4.4 Decimal

Base = decimal

Scale = fixed

Mode = real

Precision = 14 digit, signed integer

The alphanumeric representation of decimal digits uses the USASCII code extended to eight bits. The high order 4 bits are designated the zone and are 0101 for decimal digits. The low order 4 bits are the binary encoding 0000 - 1001 of the digits 0 - 9 respectively.

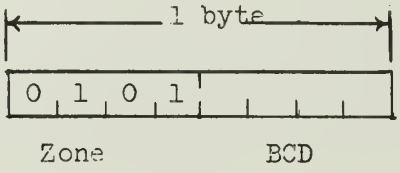
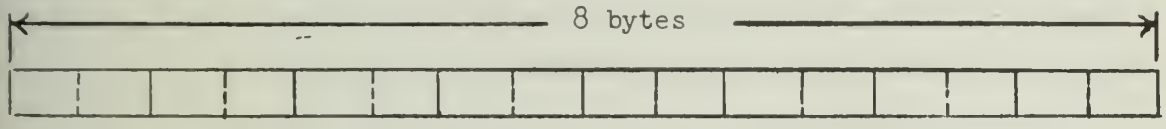


Figure 1.4.4.1 - USASCII Digit Format

A plus sign (+) is represented as 1011 and a minus sign (-) as 1101 in the right half of the left-most byte of the number, i.e. in the half byte designated "S" in Figure 1.4.4.2.*

*Definition of USASCII-8 taken from IBM System/360 Principles of Operations, File S360-01, Form A22-6821-7, p. 150.1.

Decimal operands, i.e., decimal numbers to be sent to an arithmetic unit, must be in a packed or unzoned format with two digits rather than one digit per byte. A decimal number consists of a double word subdivided into BCD digits and a 4 bit sign as shown below:



X S Two BCD digits per byte

X = Not used

S = Sign

\+ = 1011

\- = 1101

Number Type Code = 11

Figure 1.4.4.2 Decimal Number Format

Alphanumeric, or zoned format, may be converted to the above format by use of the PACK instruction of the TP.

The decimal number zero may be either plus or minus, but in either case an Equal Zero (EQ) indicator will be turned on when it is tested with a Test Algebraic (TA) instruction.

1.5 Instructions Executed by Arithmetic Units

This section describes the arithmetic instructions. The arithmetic instructions which are executed in a Taxicrinic Processor are included in Section 2.2.9. of the Programming Manual. An asterisk(*) by a name denotes an order which will not be available in the initial version of the Illiac III Arithmetic Units. Provisions are being made however for their eventual incorporation, either by additional hardware or as programmed macro-instructions. The following conventions are used in this section.

a. "Fixed" includes both short and long fixed point format unless otherwise noted.

b. Δ indicates the location of the Operand Stack Pointer (ØSP).

c. The abbreviations used for Indicators are as follows:

ØV = Overflow
LS = Loss of Significance
GT = Greater Than
EQ = Equal
LT = Less Than
FM = Flag Match
UN = Underflow
ID = Invalid Decimal Data

d. The flags of results of unary operations other than number type conversions are unchanged.

e. The "flag" convention for binary and multi-cycle arithmetic is of the form:

$$F(W) \leftarrow F(X)$$

where F = "the flags of"
X = the operand next to the top operand
W = the result produced
← = "replaced by"

f. "Flag" convention for the conversion instructions is of the form:

$F'_{i-j} \leftarrow F_{m-n}$,

where the flags within a number type are numbered consecutively 0, 1, ..., 7 from left to right.

F_{m-n} = the mth through nth flag of the original, unconverted number.

F'_{i-j} = the ith through jth flag of the resultant converted number.

← "replaced by"

g. Note that when an arithmetic computational error occurs, the flag setting of the result is not a copy of the flags of the operand, but is rather an indication of the type error which has occurred. (See Section 1.5.10).

Arithmetic Indicators include two types of indicators; comparison indicators and computational condition indicators.

Since all except fixed point comparisons (CPRA) are executed in an arithmetic unit, provisions have been included to return the results of the comparison (GT, LT, EQ, FM) to the Taxicrinic Processor. This transfer is accomplished by returning a floating point zero with the flags set to indicate the result of the comparison.

When this pseudo-result, zero, is received by the Taxicrinic Processor, the flags indicating the results of the comparison set indicator flipflops which may be tested by a subsequent instruction. For fixed point comparisons and test algebraic (TA), both of which are executed solely in a taxicrinic processor, the indicator flipflops are set directly.

The comparison indicators are designated as follows:

EQ = Equal Zero
GT = Greater Than
LT = Less Than
FM = Flags Match

The flags of a result which set these indicators are called comparison indicator flags, and are described in Table 1.5.1.

The comparison indicator flags are assigned as follows:

	Flag of Byte Number
GT	5
EQ	6
LT	7
FM	8

Byte numbering is 0 to 7, left to right.

Table 1.5.2 is a summary of the arithmetic unit order code.

TABLE 1.5.1 COMPARISON INDICATORS

INDICATOR AND DESCRIPTION	ORDERS IN WHICH INDICATOR MAY OCCUR	*PSEUDO RESULT RETURNED FROM AU
GT - Greater Than		
A > O	TA	None
A > B	CPRA	GT = 1 ZERØ
EQ - Equal		
A = O	TA	None
A = B	CPRA	EQ = 1 ZERØ
LT - Less Than		
A < O	TA	None
A < B	CPRA	LT = 1 ZERØ
FM - Flags Match		
Flags of A = O, FM = 1	TA	None
Flags of A ≠ O, FM = O		
Flags of A = Flags of B, FM = 1	CPRA	FM = 1 ZERØ
Flags of A ≠ Flags of B, FM = O		

*An arithmetic unit is used only for Decimal and Floating CPRA. TA is executed in the taxicrinic processor for all number types. In all cases the operands in the ØS are not changed.

TABLE 1.5.2 ILLIAC III ARITHMETIC UNIT ORDER CODE

ORDER		NUMBER TYPE			
Mnemonic	Instruction Variant	Short Fixed	Long Fixed	Floating	Decimal
(None)	0000	*	*	*	*
CVL	0001	TP	*	10	11
CVF	0010	00	01	*	11
CVD	0011	00	01	10	*
NEG	0100	TP	TP	TP	TP
ABS	0101	TP	TP	TP	TP
MNS	0110	TP	TP	TP	TP
TA	0111	TP	TP	TP	TP
ADD	1000	TP	TP	10	*
(None)	1001	*	*	*	*
SUB	1010	TP	TP	10	*
CPRA	1011	TP	TP	10	*
MPY	1100	00	01	10	*
POLY	1101	*	*	10	*
DIV	1110	00	01	10	*
(None)#	1111	*	*	*	*

*Not defined. No operation will take place.

TP - This operation performed in Taxicrinic Processor.

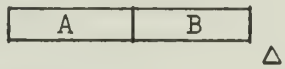
- IV = 1111 is used to indicate the final coefficient in a POLY order.

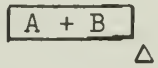
.5.1 Add

ADD

1	0	1	0	0	0	N	T

Add the top two numbers in the ØS, decrement the ØSP by CS (cell size) and load the sum into the new top of stack position.

Before ADD 

After ADD 

Flags: F(A + B) ← F(A)

Fixed: Indicators: ØV (Executed in TP)

Floating: Indicators: ØV, UN, LS

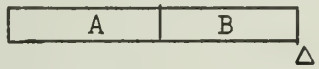
*Decimal: Indicators: ØV, ID

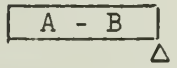
.5.2 Subtract

SUB

1	0	1	0	1	0	N	T

Subtract the top number in ØS from the next-to-top number, decrement the ØSP by CS and place the difference in the new top of stack position.

Before SUB 

After SUB 

Flags: F(A - B) ← F(A)

Fixed: Indicators: ØV (Executed in TP)

Floating: Indicators: ØV, UN, LS

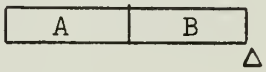
*Decimal: Indicators: ØV, ID

1.5.3 Multiply

MPY | 1 0 | 1 1 0 0 | N T |

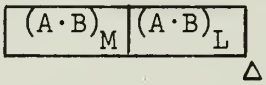
Multiply the next-to-top number (multiplicand) by the top number (multiplier) in the ØS.

Before MPY



Fixed: The most significant part of the product $(A \cdot B)_M$ replaces the multiplicand; the least significant part $(A \cdot B)_L$ replaces the multiplier. Cell size of result is twice CS of operands.

After MPY

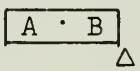


Flags: $F[(A \cdot B)_M] \leftarrow F[A]$, $F[(A \cdot B)_L] = 0$

Indicators: ØV

Floating: The normalized result replaces the multiplicand, and the ØSP is decremented by CS.

After MPY

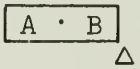


Flags: $F[A \cdot B] \leftarrow F[A]$

Indicators: ØV, UN

*Decimal: The unnormalized result replaces the multiplicand, and the ØSP is decremented by CS. As both operands and the result are double words, the sum of the number of significant digits in the operands must be ≤ 14 to prevent overflow.

After MPY



Flags: $F[A \cdot B] \leftarrow F[A]$

Indicators: ØV, ID

*A double length result is returned for fixed multiply so as to permit fixed point numbers to be interpreted as either a fraction or integer. Overflow of the single length boundary is checked and a flag is set if it occurs, however, a Bogus Result interrupt is not generated.

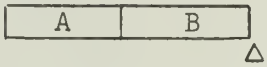
1.5.4 Divide

DIV

1	0	1	1	1	0	N	T

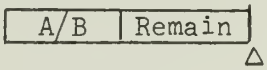
Divide the next-to-top number (dividend) by the top number (divisor) in the ØS.

Before DIV



Fixed: The quotient replaces the dividend and the remainder replaces the divisor.

After DIV

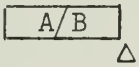


Flags: F(A/B) ← F(A), F(Remainder) = 0

Indicators: ØV

Floating: The quotient replaces the dividend, and the ØSP is decremented by CS.

After DIV

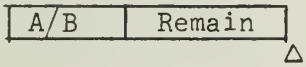


Flags: F(A/B) ← F(A), F(Remainder) = 0

Indicators: ØV, UN

*Decimal: The quotient replaces the dividend and the remainder replaces the divisor.

After DIV



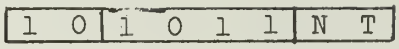
Flags: F(A/B) ← F(A), F(Remainder) = 0

Indicators: ØV, ID

Remainder has same sign as dividend (A-Op) except zero is always positive.

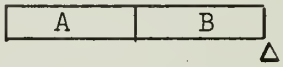
1.5.5 Compare Algebraically

CPRA

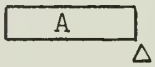


Compare algebraically the next-to-top number in ØS with the top number in the ØS. Set GT, LT, EQ Indicators.
Compare flags for match or no match and set appropriate indicator.
Decrement ØSP by CS.
If A - B > 0, set GT
If A - B = 0, set EQ
If A - B < 0, set LT
If flags of A match flags of B, set FM

Before CPRA



After CPRA



Flags: Unchanged

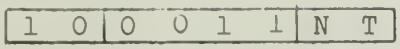
Fixed: Indicators: GT, LT, EQ, FM (Executed in TP)

Floating: Indicators: GT, LT, EQ, FM,

*Decimal: Indicators: GT, LT, EQ, FM, ID,

1.5.6 Convert to Decimal

CVD



Convert the specified number on top of ØS into a packed (2 BCD/byte) double word decimal number.

Short Fixed: Flags: $F'_{0-1} \leftarrow F_{0-1}$

$F'_{2-7} = 0$

Indicators: None

Long Fixed: Flags: $F'_{0-3} \leftarrow F_{0-3}$

$F'_{4-7} = 0$

Indicators: None

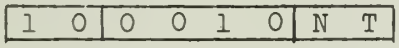
Floating: Flags: $F'_{0-7} \leftarrow F_{0-7}$

Indicators: OV

Decimal: NØP

1.5.7 Convert to Floating Point

CVF



Convert the specified number at top of ØS into a normalized floating point number.

Short Fixed: Flags: $F'_{0-1} \leftarrow F_{0-1}$

$F'_{2-7} = 0$

Indicators: None

Long Fixed: Flags: $F'_{0-3} \leftarrow F_{0-3}$

$F'_{4-7} = 0$

Indicators: None

Floating: NØP

Decimal: Flags: $F'_{0-7} \leftarrow F_{0-7}$

Indicators: ID

1.5.8 Convert to Long Fixed Point

CVL

1	0	0	0	0	1	N	T

Convert the specified number at top of ØS into a long fixed point number.

Short Fixed: Flags: $F'_{0-1} \leftarrow F_{0-1}$ (Executed in the TP)

$F'_{2-3} = 0$

Indicators: None

Long Fixed: NØP

Floating: Flags: $F'_{0-3} \leftarrow F_{0-3}$

F_{4-7} are lost

Indicators: ØV

Decimal: Same as above

Indicators: ØV, ID

1.5.9 Polynomial Evaluation

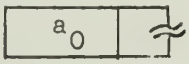
POLY | 1 0 | 1 1 | 0 1 | N | T |

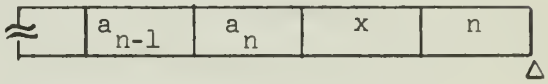
The polynomial of the form

$a_n x^n + a_{n-1} x^{n-1} + \ldots a_1 x + a_0$ is evaluated.

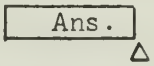
To specify the polynomial, the coefficients a_0 through a_n are pushed into the ØS in that order. The value of x is then pushed in followed by the degree n, a short fixed point number.

Before POLY





After POLY



Flags: F(Result) ← F(X)

*Fixed: Indicators: ØV,

Floating: Indicators: ØV, UN, LS

*Decimal: Indicators: ØV, LS, ID

1.6 Exceptional Conditions for Arithmetic Instructions

1.6.1 General

When a computational condition such as an overflow occurs in an arithmetic unit, this information must be returned to the taxicrinic processor. As with comparison operations (Section 1.5.2), the flags of a result are used to return this condition information, and to set indicator flipflops which may be tested with subsequent instructions. The bogus result produced is returned to the Taxicrinic Processor, not as a copy of the flags of an operand, but rather as an indication of the condition which has occurred. The fact that an error has occurred is included with the Exchange Net transmission in the control byte.

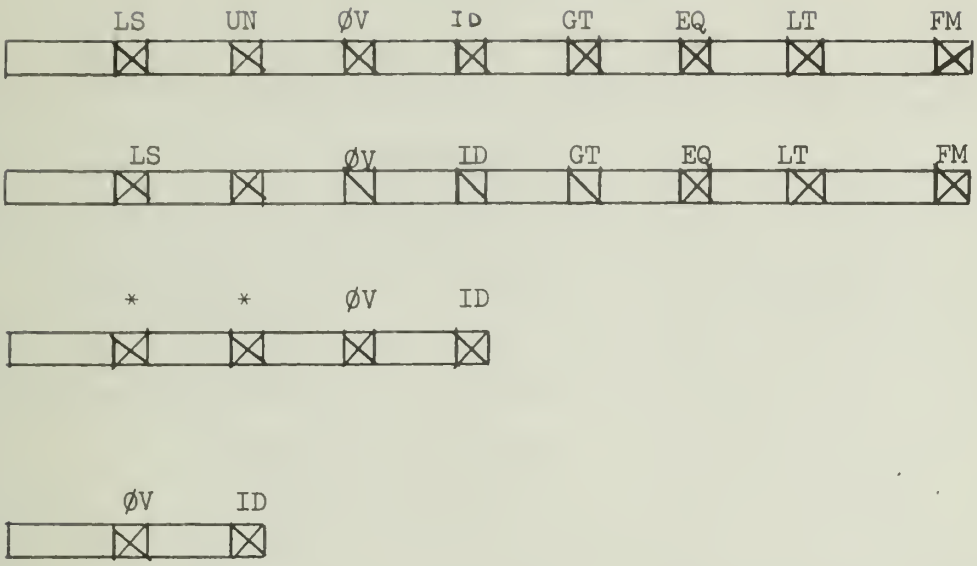
The bogus result with the appropriate computational condition flag(s) set is pushed into the ØS as if it were a correct result. An interrupt then takes place except for an overflow for fixed point multiply. (See Sec. 1.5.3 - 1

For the reader familiar with PL/I, Table 1.6.1.2 describes the analogy between the computational conditions of PL/I* and those implemented in the hardware of Illiac III. Although there is not a one-to-one correspondence between the two versions, both yield approximately the same information if the instruction and number type are known.

The following parts in this section describe the Illiac III computational condition indicators and illustrate the disposition of bogus results. The assignment of computational condition flags is illustrated for the various number types in Figure 1.6.1.1. The comparison indicator flags (Section 1.5)are also shown for the sake of completeness.

---

*"IBM Operating System/360, PL/I - Language Specifications," File No. S360-29, Form C28-6571-4, p. 162.



= Flag of Byte (Bit #9)

* = Not Used in this Number Type

Figure 1.6.1.1 Flag Bit Designation for Arithmetic Indicators

Table 1.6.1.2 Correspondence between Computational Conditions of PL/I and Those of Illiac III

PL/I	ILLIAC III		
	Instruction Variant	Number Type	Computational Condition Indicator
CONVERSION	CVF or CVL	Decimal	Invalid Decimal Data
	CVD or CVL	Floating	Overflow
	CVL	Decimal	Overflow
FIXEDOVERFLOW	ADD, SUB, ABS, MPY, DIV, POLY	Long or Short Fixed or Decimal	Overflow
OVERFLOW	ADD, SUB, MPY, DIV, POLY,	Floating	Overflow
SIZE	---	---	No hardware implementation.
UNDERFLOW	ADD, SUB, MPY DIV, POLY	Floating	Underflow
ZERODIVIDE	DIV	Any	Overflow

Table 1.6.2.1 - OVERFLOW (OV)

ORDER IN WHICH CONDITION MAY OCCUR	DESCRIPTION OF CONDITION	RESULT IN STACK
ADD, SUB		
S. Fixed (also in CPRA)	Magnitude of result exceeds $(2^{15}-1)$.	Low order 16 bits of the result.
L. Fixed (also in CPRA)	Magnitude of result exceeds $(2^{31}-1)$.	Low order 32 bits of the result.
Floating	The exponent of the normalized result exceeds 63 and the result fraction is not zero.	Fraction is that computed and correctly normalized. Sign of fraction is correct. Exponent = +63.
Decimal	Magnitude of result exceeds 99999999999999 (14, 9's).	Low order 14 digits of the result.
MPY, POLY		
S. Fixed	Magnitude of result exceeds $(2^{15}-1)$.	Full-word correct result with OV flag set.

Table 1.6.2.1 - Overflow (OV) (Continued)

Order in Which Condition May Occur	Description of Condition	Result in Stack
L. Fixed	Magnitude of result exceeds $(2^{31}-1)$.	Double word (integer) correct result with OV flag set.
Floating	The exponent of the normalized result exceeds 63 and the result fraction is not zero.	Fraction is that computed and correctly normalized. Sign of fraction is correct. Exponent = +63.
Decimal	Magnitude of result exceeds 14, 9's.	Low order 14 digits of the result.
DIV		
S. Fixed L. Fixed	Division by zero.	Largest number representable. Sign of result is sign of dividend. Remainder is 0.
Floating	Division by zero.	Fraction and exponent are all 1's. Sign of fraction is the sign of the dividend.
Decimal	Division by zero.	Largest number representable.

Table 1.6.2.1 - Overflow (OV) (Continued)

Order in Which Condition May Occur	Description of Condition	Result in Stack
ABS, NEG		
S. Fixed	Attempt to negate the most negative number representable.	The integer +1.
CVD		
Floating	The magnitude of the converted number exceeds the range of the new number type.	Result is the converted * number with missing high order digits which have overflowed.
CVL		
Floating Decimal	The magnitude of the converted number exceeds the range of the new number type.	Result is the converted number with missing high order bits which have overflowed.

* This is true only if Exponent of Floating Operand is ≤ 14. Otherwise "double" overflow occurs and error is compounded.

1.6.3 Underflow (UN)

Table 1.6.3.1 - UNDERFLOW (UN)

ORDER IN WHICH CONDITION MAY OCCUR	DESCRIPTION OF CONDITION	RESULT IN STACK
ADD SUB MPY DIV POLY Floating Only	The exponent of the normalized result is less than -64 and the result fraction is not zero.	Fraction is that computed and correctly normalized. Sign of fraction is correct. Exponent is -64.

1.6.4 Invalid Decimal Data (ID)

Table 1.6.4.1 - INVALID DECIMAL DATA (ID)

ORDER IN WHICH CONDITION MAY OCCUR	DESCRIPTION OF CONDITION	RESULT IN STACK
Any Decimal Order Except NEG, ABS, MNS, TA.	A sign or digit code of an operand is incorrect.	Result is the contents of the AU accumulator when the decoding error was detected. No ID check is made for unary operations except CVF and CVL.

1.6.5 Loss of Significance (LS)

Table 1.6.5.1 - LOSS OF SIGNIFICANCE (LS)

ORDER IN WHICH CONDITION MAY OCCUR	DESCRIPTION OF CONDITION	RESULT IN STACK
ADD SUB POLY Floating Only	Fraction of result is 0. Exponent of result $\neq -64$. For example- when two equal numbers are subtracted.	True zero with LS flag set.

2. INTERNAL STATIC DESCRIPTION

2.1 General

2.1.1 Overall Block Diagram

Figure 2.1.1.1 is a block diagram of an Illiac III arithmetic unit. The conventions used in this figure are as follows:

1) Functional sub-blocks are denoted by rectangles. Inside each box is the name of the block followed by a list of the names of signals which control it.

2) The lines between boxes denote data buses.

3) Selector signal names are of the form F X T, where

   F is the name of the register from which the data is transferred.
   X = D if the transfer is direct, i.e. without shifting.
   X = R_n if the data is shifted n places to the right during the transfer.
   X = L_n if the data is shifted n places to the left during the transfer.
   T = the name of the register to which data is transferred.

4) A register name standing alone, for example, UQ, denotes the true output of all positions of the register. A subsection of a register is specified in the following form:
   <register name> np,
   where n is the number of the first byte (8 bits per byte) of the subsection and p is the number of the last byte of the subsection. Byte numbering is 0 through 7. Example: VDUH47 means V-BUS Direct to UH-Register, bytes 4 through 7.

5) If R denotes the name of a register, then RSEL denotes the output of the associated input selector.

6) If R denotes the name of a register, then LDR denotes the signal which loads the output of the associated select into the register flip-flops.

7) All selectors, registers, subtracters and shift gates are 64 bits (8 bytes) wide, except for the M-Register which is 56 bits wide.

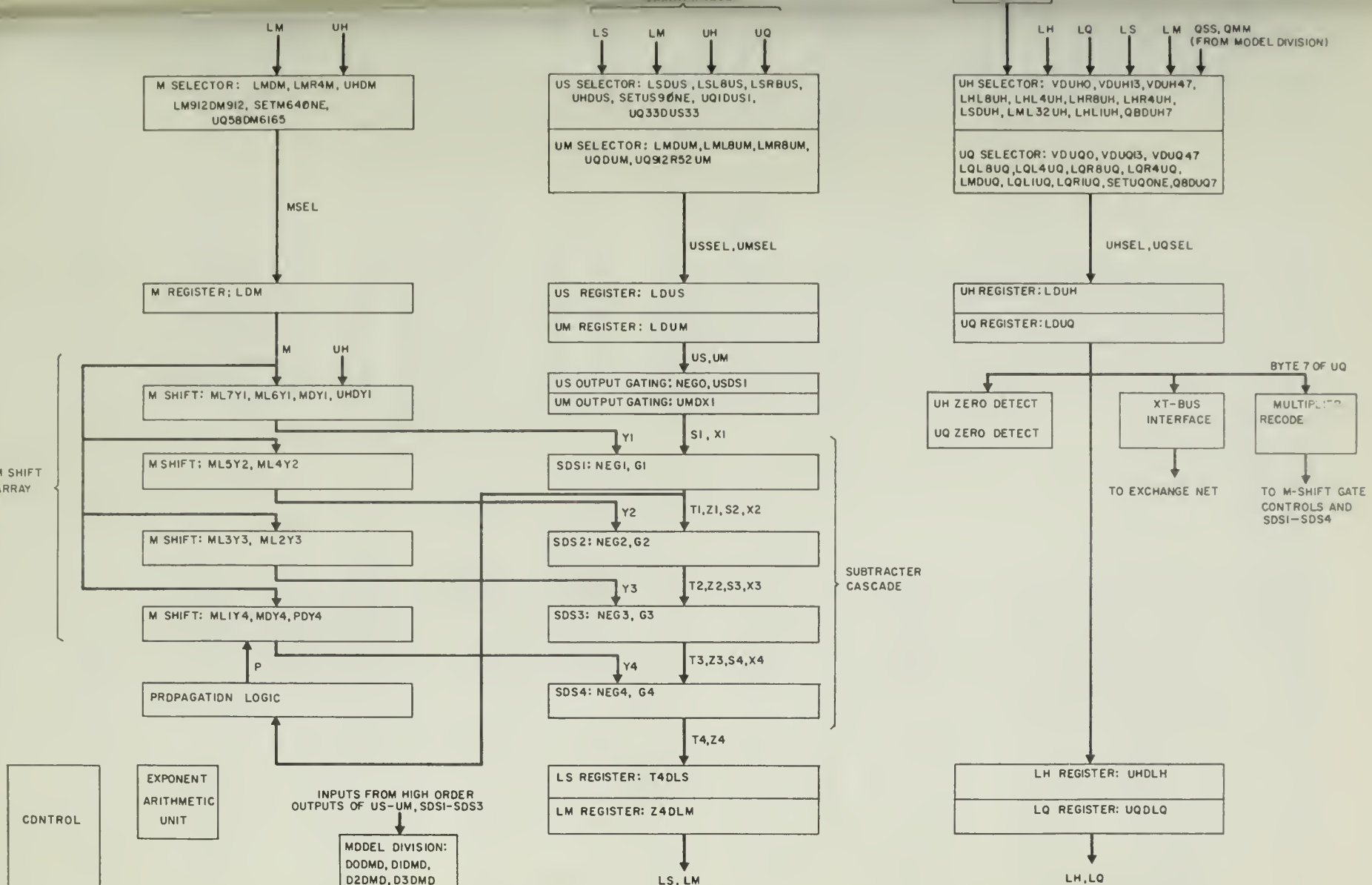


FIGURE 2.1.1.1.

ILLIAC III ARITHMETIC UNIT BLOCK DIAGRAM (VER. 2)

D.E. ATKINS 4-28-69

Table 2.1.1.1 - Summary of the Sub-Blocks of the Main Arithmetic Unit
Reference: Figure 2.1.1.1

Abbreviated Name	Name	Description
M	Multiplicand Register	Holds a number in conventional binary form to be added or subtracted from the redundantly represented number in US-UM. Connected to the Y-inputs of the SDS cascade via the shift gate array.
US	Upper Sign Register	Part of the primary rank of the double rank registers associated with the SDS array. Holds the sign bits of a binary number in SD format to be used as input to SDS1
UM	Upper Magnitude	Part of the primary rank of the double rank register associated with the SDS array. Contains the magnitude of binary numbers in SD format to be used as input to SDS1.
LS	Lower Sign Register	Secondary rank for the US register
LM	Lower Magnitude Register	Secondary rank for the UM register
UH	Upper H-Register	The primary rank of a double rank register used to hold the sign bits of a quotient and in aligning and normalizing operands.
LH	Lower H-Register	The secondary rank for the UH Register.
UQ	Upper Q-Register	The primary rank of a double rank register used to hold the magnitude bits of a quotient and in aligning and normalizing operands. Holds the multiplier during multiplication.

Abbreviated Name	Name	Description
LQ	Lower Q-Register	The secondary rank for UQ.
V-BUS	In Bus. (The "V" denotes an arrow-head point into the system)	Provides input (to AU) interface between Exchange Net and AU. Includes cable terminators and parity checking.
XT-BUS	Exit Bus	Provides output (from AU) interface between AU and Exchange Net. Includes cable drivers and parity generation.
MR	Multiplier Recode	Recodes the lower order 9 bits of UQ-Register into signals which operate the gates of the M-Shift array during Multiplication.
SDS1-SDS4	Signed-Digit Substructed 1 through 4	Signed Digit Subtracters.
ZD	Zero Detect	Detects the presence of all zeros in the register to which it is connected.
PL	Propagation Logic	Produces the P_i bits from the T_i and Z_i outputs of S1. These P_i bits are then combined with Z_i bits in S4 to produce the assimilated result. This unit is similar to borrow look-ahead logic.
MSA	M Shift Array	Selects multiples of the contents of the M-Register to be added in SDS1, SDS2, SDS3, SDS4 of the Subtracter Cascade.
MD	Model Division	A radix four, table look-up division which generates quotient digits to be stored in UH-UQ and to control the M-Shift Array in forming full precision partial remainders.
EAU	Exponent Arithmetic Unit	Performs arithmetic on the 7 bit exponents of floating point operands.

2.1.2 Comments on the Structure

This section includes comments about the general layout of the block diagram. It is hoped that this discussion of the "big picture" will help the myriad details of subsequent sections to cohere.

For purposes of discussion, it is fruitful to recognize the functional facilities listed in Table 2.1.2.1 together with the major associated hardware as shown in the block diagram. We shall now comment on each of these facilities.

2.1.2.1 Input

Operands enter an AU through the V-BUS at the top of the block diagram. The mnemonic for V is the observation that the V is an arrowhead pointing into the unit.

Facility	Associated Hardware
1. Input	V-BUS UH-Register, UQ-Register
2. Initial and Terminal Operations	UH-LH-Register UQ-LQ-Register
3. Addition - Subtraction	US-LS-Register UM-LM-Register The SDS array SDS1, SDS2, SDS3, SD Propagation Logic
4. Generation of Multiples	M-Register M-Shift Array Multiplier Recode
5. Output	UQ-Register XT-BUS
6. Quotient Digit Generation	Model Division
7. Condition Detection	Zero Detect, Condition Detectors
8. Exponent Arithmetic	Exponent Arithmetic Unit
9. Control	Control

TABLE 2.1.2.1 - AU Functional Facilities

The contents of the V-BUS may load the M-Register, the UH-Register, or the UQ-Register. Since only four bytes of data are available on the V-BUS, the high order and low order portions of these registers are loaded in sequence. Most operations begin by loading the UH and UQ-Registers. Parity is checked at the V-BUS interface.

2.1.2.2 Initial and Terminal Operations

Due to the connections to the V-BUS and XT-BUS, and the high-speed shift capabilities, the H and Q Registers are associated with initial and terminal operations. The UH (Upper H) and UQ (Upper Q) are the primary rank of double rank registers, while the secondary ranks are LH (Lower H) and LQ (Lower Q), respectively. Operands stored in the UH and UQ may be shifted right or left by any multiple of 4 bits.

The UH-UQ-Registers are used, for example, in the initial operations of floating ADD to perform right shifts on one of the fractions until the value of exponents agree. In the terminal operations of floating ADD, they are used to normalize the fraction of the result. These registers also hold the multiplier during MPY, and the quotient during DIV.

Note that after initial operations in UH-UQ, the operands may be transferred to the US-UM-Registers or the M-Register. Likewise, when terminal operations are required, the result may be transferred from the LS-LM-Registers to the UH-UQ-Registers.

2.1.2.3 Addition-Subtraction

All addition and subtraction, except that involving exponents, is performed in the Signed-Digit Subtracter Cascade. This array consists of a cascade of four signed-digit subtracters as described in Section 2.5

and labeled SDS1 through SDS4 on the block diagram. Actually these devices will also add. Which function is performed is determined by the setting of the NEG line shown to the right of each subtracter and between US and SDS1. The accumulator for this array is the US-UM-Registers together with the secondary rank, the LS-LM-Registers.

The Signed-Digit Subtracter (SDS), as the name implies, performs arithmetic on signed-digits, i.e. each digit is composed of two bits: a sign and a magnitude. This representation is said to be in SD format. The prime advantage of this scheme is the fact that in repetitive additions, as for example in MPY, the carry propagation may be postponed until a single terminal operation. The prime disadvantage of such a scheme is the increased hardware, e.g. the requirement for an accumulator with two bits per digit. Thus the primary rank of the accumulator must consist of two registers: the UM (Upper Magnitude) and US (Upper Sign). The other input to the subtracters via the M-Shift Array is in conventional one-bit-per-digit, binary form.

Eventually the result in SD Format must be assimilated into conventional form. This assimilation requries a propagation of borrows. Note that the output of the subtracter SDS1 is connected to the PROPAGATION logic. The PROPAGATION logic generates the so-called P-bits. The P-bits are then combined in SDS4 with the number in SD format (it ripples through SDS2 and SDS3 to produce the assimilated result in LM.

Of course only one subtracter (remember, they add too) is required for an ADD or SUB. The cascade of four subtracters is justified by the requirement for high-speed multiplication.

2.1.2.4 Generation of Multiples

The complex of gates to the left of the SDS cascade comprises the M-Shift array. The input to these gates is the output of the M-Register. The outputs drive the conventional (Y) inputs to SDS1, SDS2, SDS3, SDS4.

For ADD, SUB, or CPRA, one operand is stored in the US-UM-Registers; the other in the M-Register. In this case, the only gate of the M array which is operated is MDY1. The remaining gates function during the execution of MPY and DIV. For example, in MPY the multiplicand is stored in M-Register, the multiplier in the UQ-Register. The low-order byte of the UQ is recoded and the output of the multiplier recode logic sets the appropriate M-shift array gate.

Actually multiples produced by the M array may be added or subtracted from the contents of US-UM. Which operation is performed depends upon the setting of the NEG controls shown with each signed-digit subtracter. This control is defined in detail in Section 2.5.

Table 2.1.2.4.1 further describes each M-shift array gate signal.

Gate Name	Multiple of Contents of M Produced	Output Drives
ML7Y1	128	Y1 input to SDS1
ML6Y1	64	Y1 input to SDS1
MDY1	1	Y1 input to SDS1
ML5Y2	32	Y2 input to SDS2
ML4Y2	16	Y2 input to SDS2
ML3Y3	8	Y3 input to SDS3
ML2Y3	4	Y3 input to SDS3
ML1Y4	2	Y4 input to SDS4
MDY4	1	Y4 input to SDS4
PDY4	(Output of propagation logic to Y4 input to SDS4)	

TABLE 2.1.2.4.1 - M-Shift Array Gate Description

2.1.2.5 Output

The output of an AU is from the UQ-Register. Recall that the UQ and its partner, the UH-Register, are associated with terminal operations. The output of the UQ couples to the XT-BUS interface and from there to the Exchange System. The double word contents of the UQ must be returned as a sequence of two, one-word transmissions. Parity is generated by the XT-BUS interface with the flags stored in the F-Register replaced into the results.

2.1.2.6 Quotient Digit Generation

Division in an Illiac III AU is accomplished using an advanced technique suggested by J. E. Robertson of the Department of Computer Science. The main property of this technique is that quotient bits may be formed by inspection of only the first few bits of the divisor stored in the M-Register, and the partial remainder stored in US-UM-Register. This inspection and quotient bit generation are accomplished in the so-called Model Division. The hardware in this box performs a "model" division of the first few bits of the divisor and dividend. The results of model are then used to control the M-Shift Array to form the next, full precision partial remainder.

2.1.2.7 Condition Detection

This facility includes all hardware associated with the detection of the results of comparison operations and the detection of error conditions such as overflow and underflow. It includes the Zero Detect hardware which is used to warn of impending division by zero and also, in the case of a floating point operation, to signal a zero fraction so that the exponent may be adjusted to create a true zero.

2.1.2.8 Exponent Arithmetic

Exponent arithmetic is the exclusive domain of the Exponent Arithmetic Unit (EU). The adder employed in the EU is of the conventional type, not a signed-digit subtracter.

2.1.2.9 Control

Control is built around the concept of the control point. A control point is a logic element which when entered will remain on a preselected but adjustable interval of time, then turn off and generate a start signal to the next control point. Conditional logic is permitted on both the "DO" output of the control point logic and between control point logic. These facilities permit conditional operations and conditional branching.

2.2 Registers

This section describes the main registers of an Arithmetic Unit. The registers are described functionally.

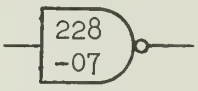
The following definitions and notation are employed:

1) The word "position" refers to a lateral section of the register capable of storing one bit. The M-Register, for example, is 56 positions wide.

2) The word "stage" refers to the direction of data, i.e., implies a series connection: a cascading.

3) Subscripts may either be written below the line, e.g., M_2 or on the line, M2. Subscripts may denote either byte or bit number. If the choice is not clear from the context, it will be explicitly stated.

4) Numbers inside logic symbols denote PC board type, as defined in the Illiac III Engineering Manual, e.g.



5) Register positions (horizontal divisions) are numbered such that position number 9 is immediately to the right of the radix point. In some cases, the M Register for example, positions 1 through 8 are not present.

6) Notice that the term "register" includes not only the flip-flop storage, but also all gating into the flip-flop and possibly gating on the outputs.

7) Signals common to many gates (e.g. selectors and load) are usually subdivided into several signals, each operating a subgroup of bytes of the register or selector.

2.2.1 M-Register

2.2.1.1 General

The M-Register consists of input selectors, 56 positions of storage, and inverters on the output to supply additional drive to the M-shift array. The primary function of this register is to store a 7-byte number which drives the M-Shift Array. The outputs of the M-Shift Array drive the "Y" inputs to the four signed-digit subtractors. The M-Register therefore holds any operand which is to be combined with the contents of the main accumulator, US-UM.

The primary inputs to the M-Register Selector are the direct outputs of the UH and LM Registers and the outputs of LM-Register shifted right four bit positions. There are also special inputs at each end of the registers. These are defined in Section 2.2.1.3. The outputs of the M-Selector, $MSEL_i$ (i = 9 to 64), to be loaded into the flip-flops under control of LDM (Load M).

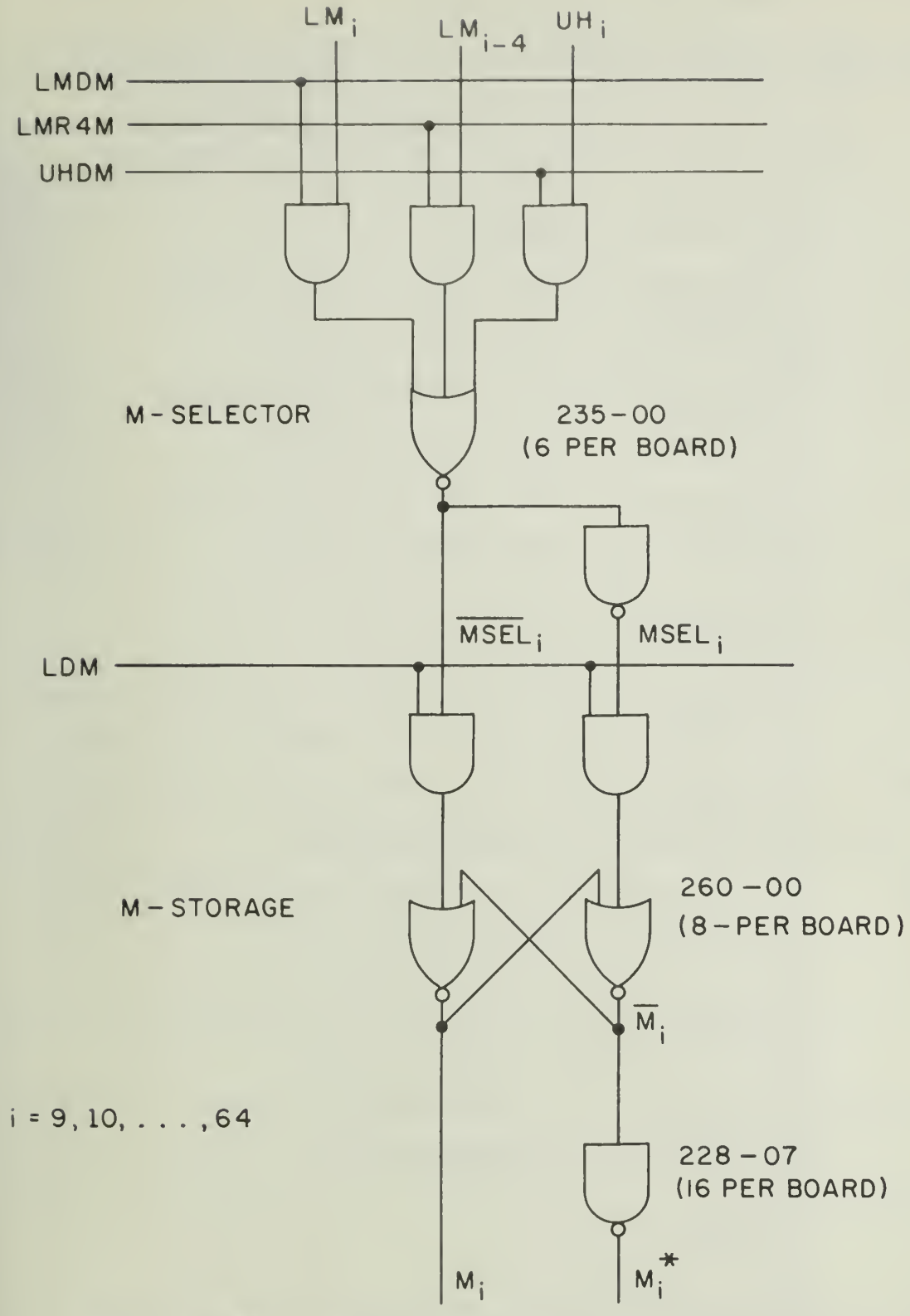
Note that although the M-Register consists of only 56 positions, the positions are designated 9 through 64. The M-Register may therefore be thought of as a full 8-byte register, as are the other main registers of the AU, but with the first byte not actually implemented.

2.2.1.2 Implementation

The bulk of the M-Selector is implemented with the 235-00 card, although at the high order end 241-00 boards are used in a dot-OR configuration. The signals LM912DM912, SETM640NE, and UQ58DM6165 are required for conversion orders and although not shown in Figure 2.2.1.2.1, are precisely defined in PL/1 definitions in the next section.

Figure 2.2.1.2.1 illustrates the logic of a typical position of the M-Register.

The relevant detailed logic drawings are 210-01, -02, -03, -04, -05 and -06.



NOTE: M_i^* DENOTES A SIGNAL LOGICALLY EQUIVALENT TO M_i BUT ELECTRICALLY DISTINCT.

Figure 2.2.1.2.1 - Typical Position of M-Register

```
/* PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO M-REGISTER */
/* RELEVANT DRAWING NUMBERS: 221-01,-02,-03,-04,-05,-06 */
        DECLARE M BIT(64);/* TRUE OUTPUT OF M-REGISTER.
                             POSITIONS 1-8 NOT IMPLEMENTED.*/
        DECLARE MSEL BIT(64); /* TRUE OUTPUT OF M-SELECT.
                             POSITIONS 1-8 NOT IMPLEMENTED. */
LDM:      PROCEDURE;
          CALL LMD13; CALL LMD47;
          END;
LDM13:    /*LOAD OUTPUT OF MSEL 13-32 INTO M 13-32*/
          PROCEDURE;
          SUBSTR(M,13,20)=SUBSTR(MSEL,13,20);
          END;
LDM45:    /*LOAD OUTPUT OF MSEL 33-48 INTO M 33-48*/
          PROCEDURE;
          SUBSTR(M,33,16)=SUBSTR(MSEL,33,16);
          END;
LDM67:    /*LOAD OUTPUT OF MSEL 49-64 INTO M 49-64*/
          PROCEDURE;
          SUBSTR(M,49,16)=SUBSTR(MSEL,49,16);
          END;
LDM912:   /*LOAD OUTPUT OF MSEL 9-12 INTO M 9-12*/
          PROCEDURE;
          SUBSTR(M,9,4)=SUBSTR(MSEL,9,4);
          END;
LMDM:     PROCEDURE;
          CALL LDM912; CALL LDM13; CALL LDM45; CALL LDM67;
          END;
LMDM13:   /*LM OUTPUT BYTES 1-3 TO MSEL OUTPUT BYTES 1-3*/
          PROCEDURE;
          SUBSTR(MSEL,9,24)=SUBSTR(LM,9,24);
          END;
LMDM47:   /*LM OUTPUT BYTES 4-7 TO MSEL OUTPUT BYTES 4-7*/
          PROCEDURE;
          SUBSTR(MSEL,33,32)=SUBSTR(LM,33,32);
          END;
LMR4M:    /*LM OUTPUT 5-60 TO MSEL OUTPUT 9-64*/
          PROCEDURE;
          SUBSTR(MSEL,9,56)=SUBSTR(LM,5,56);
          END;
LM912DM912:/*LM OUTPUT 9-12 TO MSEL OUTPUT 9-12*/
          PROCEDURE;
          SUBSTR(MSEL,9,4)=SUBSTR(LM,9,4);
          END;
SETM64ONE: /*MSEL 64 = '1'B */
          PROCEDURE;
          SUBSTR(MSEL,64,1)='1'B;
          END;
UHDM:     PROCEDURE;
          CALL UHDM13; CALL UHDM47;
          END;
UHDM13:   /*UH OUTPUT BYTES 1-3 TO MSEL OUTPUT BYTES 1-3*/
          PROCEDURE;
          SUBSTR(MSEL,9,24)=SUBSTR(UH,9,24);
          END;
UHDM47:   /*UH OUTPUT BYTES 4-7 TO MSEL OUTPUT BYTES 4-7*/
          PROCEDURE;
```

```
            SUBSTR(MSEL,33,32)=SUBSTR(UH,33,32);
            END;
UQ58DM6165:/*UQ OUTPUT 5-8 TO MSEL OUTPUT 61-65*/
            PROCEDURE;
            SUBSTR(MSEL,61,4)=SUBSTR(UQ,5,4);
            END;
```

2.2.2 US-Register

2.2.2.1 General

The subtracters are members of a class of borrow-save units and thus require the redundant representation of one operand. The main accumulator, both primary and secondary rank, must provide two bits of storage per digit: one bit for the magnitude of the digit, the other bit for the sign. The US-Register (Upper Sign-Register) is an 8-byte register which stores the sign bits of the primary accumulator. The UM-Register (Section 2.2.3) stores the corresponding magnitude bits.

The US-Register complex consists of input selectors, flip-flop storage, and output gating which under control of the signal NEGO, will supply either the true or complement output of the US flip-flops to the input of the first signed-digit subtractor (SDS-1).

The inputs supplied by the selector, and their primary use are shown below:

Signal Name	Description	Primary Use
UHDUS	Select true outputs of UH-Register direct (i.e. without shifting).	Initial loading of primary accumulator from UH-Register, which serves as input buffer.
LSL8US	Select true output of LS-Register shifted left 8 bits.	Iterative portion of division.
LSDUS	Select true output of LS-Register, unshifted.	Last iteration of fixed point multiplication.
LSR8US	Select true output of LS-Register shifted right 8 bits.	Iterative portion of multiplication.

The outputs of the selector, $USSEL_i$ (i = 1 to 64), are loaded into the US flip-flops under control of LDUS (Load US).

These signals and several special signals are defined more precisely in PL/1 in Section 2.2.2.3.

The outputs of the US output gates are designated Sli (i = 1 to 64). The gating of the US-Register to SDS1 is under control of the signal USDS1: the polarity under control of the signal NEGO. More precisely: $S1_i = USDS1 \cdot (NEGO \oplus US_i)$.

2.2.2.2 Implementation

The US-Selector is implemented with the 294-00 Selector card and the 228-03 NAND Card. The storage flip-flops are the 260-00. A typical position of US Register and Selector is shown in Figure 2.2.2.2.1. The US output gating is implemented with 241-00 and 228-03 cards and a typical position is shown in Figure 2.2.2.2.2.

Relevant detail drawings are as follows: 222-01, -02, -03, -04, -05, -06, -07, -08, -09, and -10.

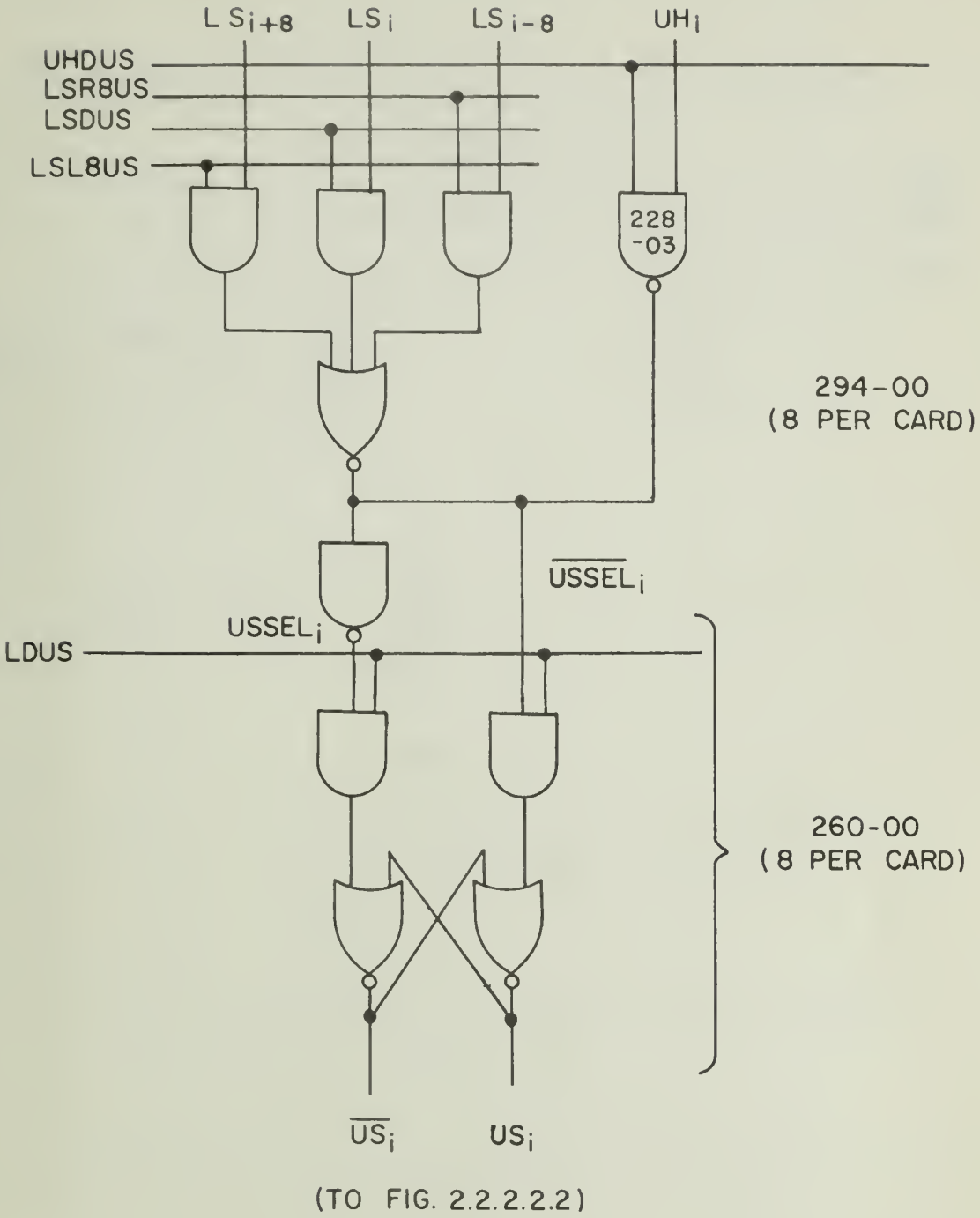


Figure 2.2.2.2.1 - Typical Position of the US Register and Selectors

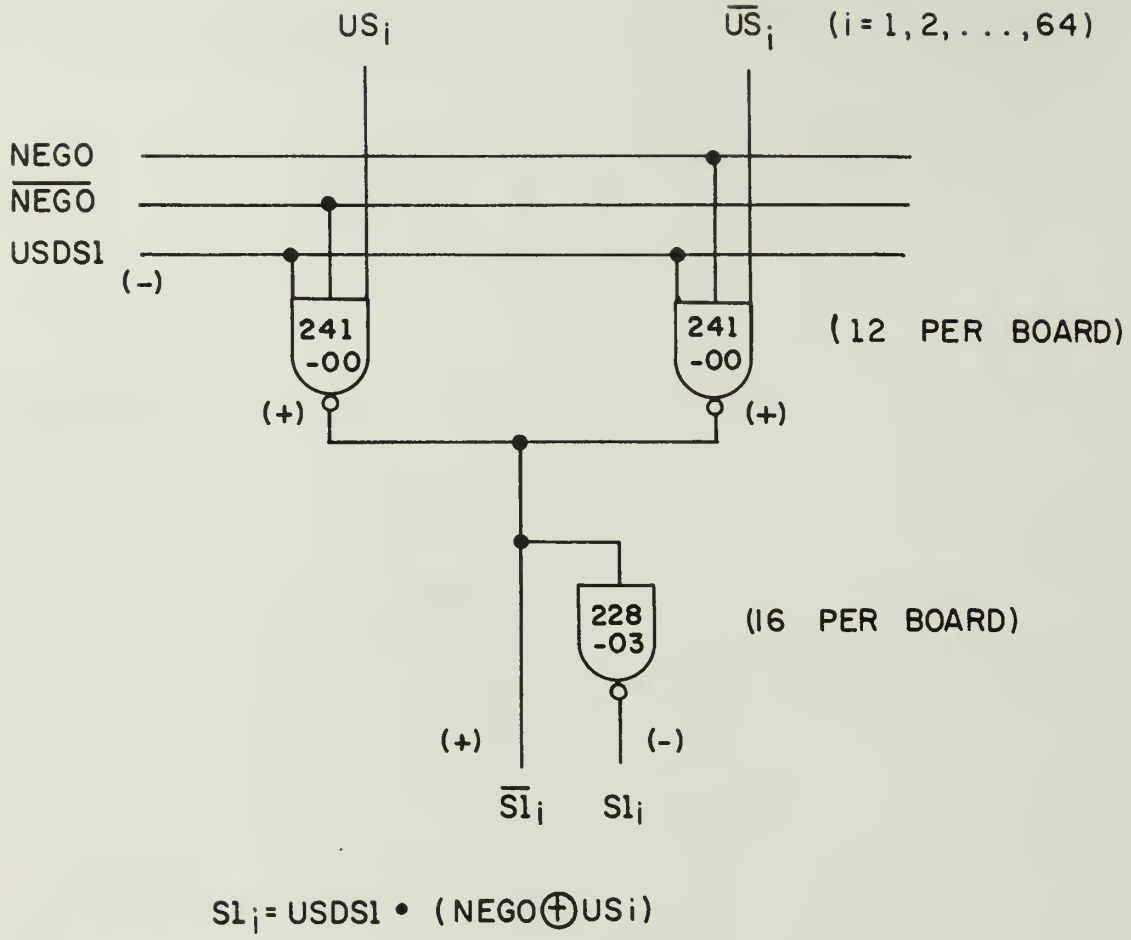


Figure 2.2.2.2.2 - Typical Position of US Output Gating

```
PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO US-REGISTER */
RELEVANT DRAWING NUMBERS: 221-01,-02,-03,-04,-05,-06,-07,-08
                          -09,-10*/
        DECLARE US BIT(64);/*TRUE OUTPUT OF US-REGISTER*/
        DECLARE USSEL BIT(64);/*TRUE OUTPUT OF US-SELECTOR*/
        DECLARE S1 BIT(64);/*OUTPUT OF US OUTPUT GATING AND
                             ONE INPUT TO SDS-1*/
        DECLARE NEGO_S BIT(64);/*NEGO_STRING*/
US:     PROCEDURE;
        CALL LDUS01; CALL LDUS23; CALL LDUS45; CALL LDUS67;
        END;
US01:   /*USSEL BYTES 0-1 TO US BYTES 0-1*/
        PROCEDURE;
        SUBSTR(US,1,16)=SUBSTR(USSEL,1,16);
        END;
US23:   /*USSEL BYTES 2-3 TO US BYTES 2-3*/
        PROCEDURE;
        SUBSTR(US,17,16)=SUBSTR(USSEL,17,16);
        END;
US45:   /*USSEL BYTES 4-5  TO US BYTES 4-5*/
        PROCEDURE;
        SUBSTR(US,33,16)=SUBSTR(USSEL,33,16);
        END;
US67:   /*USSEL BYTES 6-7  TO US BYTES 6-7*/
        PROCEDURE;
        SUBSTR(US,49,16)=SUBSTR(USSEL,49,16);
        END;
LUS:    PROCEDURE;
        CALL LSLUS03; CALL LSLUS47;
        END;
LUS03:  /*LS BYTES 0-3 TO USSEL BYTES 0-3*/
        PROCEDURE;
        SUBSTR(USSEL,1,32)=SUBSTR(LS,1,32);
        END;
LUS47:  /*LS BYTES 4-7 TO USSEL BYTES 4-7*/
        PROCEDURE;
        SUBSTR(USSEL,33,32)=SUBSTR(LS,33,32);
        END;
L8US:   PROCEDURE;
        CALL LSL8US03; CALL LSL8US47;
        END;
L8US03: /* LS BYTES 1-4 TO USSEL BYTES 0-3*/
        PROCEDURE;
        SUBSTR(USSEL,1,32)=SUBSTR(LS,9,32);
        END;
L8US47: /* LS BYTES 5-7 TO USSEL BYTES 4-6*/
        PROCEDURE;
        SUBSTR(USSEL,33,32)=SUBSTR(LS,41,24)||'00000000'B;
        END;
R8US:   PROCEDURE;
        CALL LSR8US03; CALL LSR8US47;
        END;
R8US03: /*LS BYTES 0-2 TO USSEL BYTES 0-3*/
        PROCEDURE;
        SUBSTR(USSEL,1,32)='00000000'B||SUBSTR(LS,1,24);
        END;
R8US47: /*LS BYTES 3-6 TO USSEL BYTES 4-7*/
```

```
            PROCEDURE;
            SUBSTR(USSEL,33,32)=SUBSTR(LS,25,32);
            END;
NEGO:       PROCEDURE;
            CALL NEGO03; CALL NEGO47;
            END;
NEGO03:     /*NEGATION CONTROL FOR BYTES 0-3 OF US OUTPUT*/
            PROCEDURE;
            SUBSTR(NEGO_S,1,32)=(32)'1'B;
            END;
NEGO47:     /*NEGATION CONTROL FOR BYTES 4-7 OF US OUTPUT*/
            PROCEDURE;
            SUBSTR(NEGO_S,33,32)=(32)'1'B;
            END;
SETUS9ONE:  /*USSEL BIT 9 SET TO '1'B*/
            PROCEDURE;
            SUBSTR(USSEL,9,1)='1'B;
            END;
UHDUS:      PROCEDURE;
            CALL UHDUS02; CALL UHDUS35; CALL UHDUS67;
            END;
UHDUS02:    /*UH BYTES 0-2 TO USSEL BYTES 0-2*/
            PROCEDURE;
            SUBSTR(USSEL,1,24)=SUBSTR(UH,1,24);
            END;
UHDUS35:    /*UH BYTES 3-5 TO USSEL BYTES 3-5*/
            PROCEDURE;
            SUBSTR(US,25,24)=SUBSTR(UH,25,24);
            END;
UHDUS67:    /*UH BYTES 6-7 TO USSEL BYTES 6-7*/
            PROCEDURE;
            SUBSTR(US,49,16)=SUBSTR(UH,49,16);
            END;
UQ1DUS1:    /*UQ BIT 1 TO USSEL BIT 1*/
            PROCEDURE;
            SUBSTR(USSEL,1,1)=SUBSTR(UQ,1,1);
            END;
UQ33DUS33:  /*UQ BIT 33 TO USSEL BIT 33*/
            PROCEDURE;
            SUBSTR(US,33,1)=SUBSTR(UQ,33,1);
            END;
USDS1:      PROCEDURE;
            CALL USDS102; CALL USDS135; CALL USDS167;
            END;
USDS102:    PROCEDURE;
            SUBSTR(S1,1,24)=SUBSTR(US,1,24)&¬SUBSTR(NEG_S,1,24)|
                           ¬SUBSTR(US,1,24)& SUBSTR(NEG_S,1,24);
            END;
USDS135:    PROCEDURE;
            SUBSTR(S1,25,24)=SUBSTR(US,25,24)&¬SUBSTR(NEG_S,25,24)|
                            ¬SUBSTR(US,25,24)& SUBSTR(NEG_S,25,24);
            END;
USDS167:    PROCEDURE;
            SUBSTR(US,49,16)=SUBSTR(US,49,16)&¬SUBSTR(NEG_S,49,16)|
                            ¬SUBSTR(US,49,16)& SUBSTR(NEG_S,49,16);
            END;
```

2.2.3 UM-Register

2.2.3.1 General

The UM-Register (Upper Magnitude-Register) is an 8-byte register which stores the magnitude bits of the primary accumulator. The US-Register (see Section 2.2.2) stores the corresponding sign bits.

The UM-Register complex consists of input selectors, flip-flop storage and output gating which supplies the magnitude bits to the magnitude inputs of the first signed-digit subtracter (SDS-1).

The inputs to the register supplied by the selector, and their primary uses, are shown below:

Signal Name	Description	Primary Use
UQDUM	Select true outputs of UQ register direct to UM (i.e. without shifting).	Initial loading of primary accumulator from UQ-Register which serves as I/O buffer for AU.
LML8UM	Select true outputs of LM-Register shifted left 8 bits.	Iterative portion of division.
LMDUM	Select true output of LM-Register, unshifted.	Last iteration of fixed point multiplication.
LMR8UM	Select true output of LM-Register shifted right 8 bits.	Iterative portion of of multiplication.

The outputs of the selector, $UMSEL_i$ (i = 1 to 64), are loaded into the UM flip-flops under control of LDUM (Load UM).

The outputs of the UM output gates are designated Xli (i = 1 to 64). The gating of the UM-Register to SDS1 is under control of the signal UMDX1.

These signals, including their subsignals, are defined using PL/1 in Section 2.2.3.3.

2.2.3.2 Implementation

The UM-Selector is implemented with the 294-00 Selector card and the 228-03 NAND card. The flip-flops are the 260-00. A typical position of the UM-Register and Selector is shown in Figure 2.2.3.2.1. The UM output gating is implemented with 228-03 and 228-07 cards and a typical position is shown in Figure 2.2.3.2.2.

Relevant detailed drawings are as follows: 223-01, -02, -03, -04, -05, -06, -07, -08.

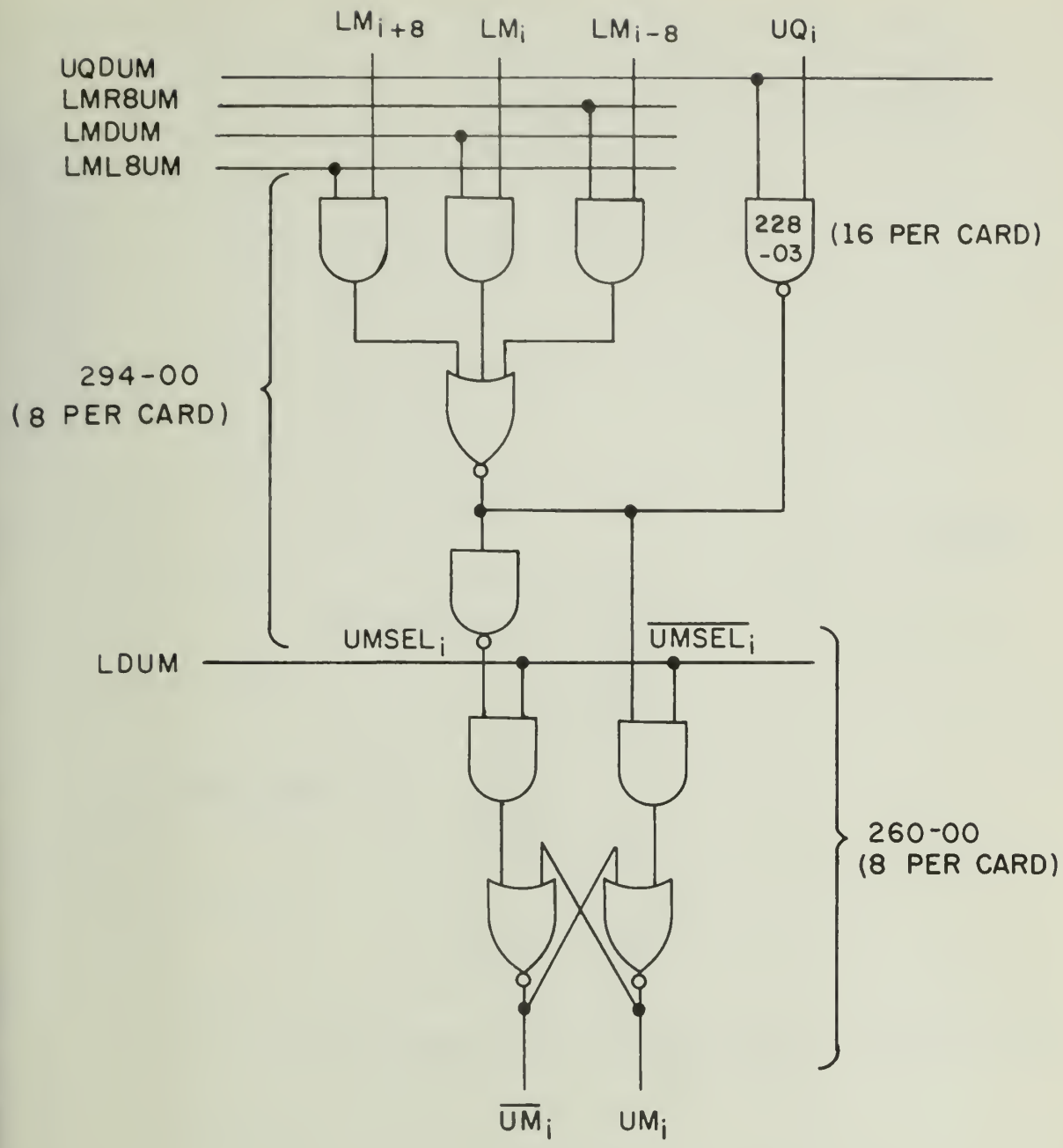


Figure 2.2.3.2.1 - Typical Position of the UM Register and Selectors

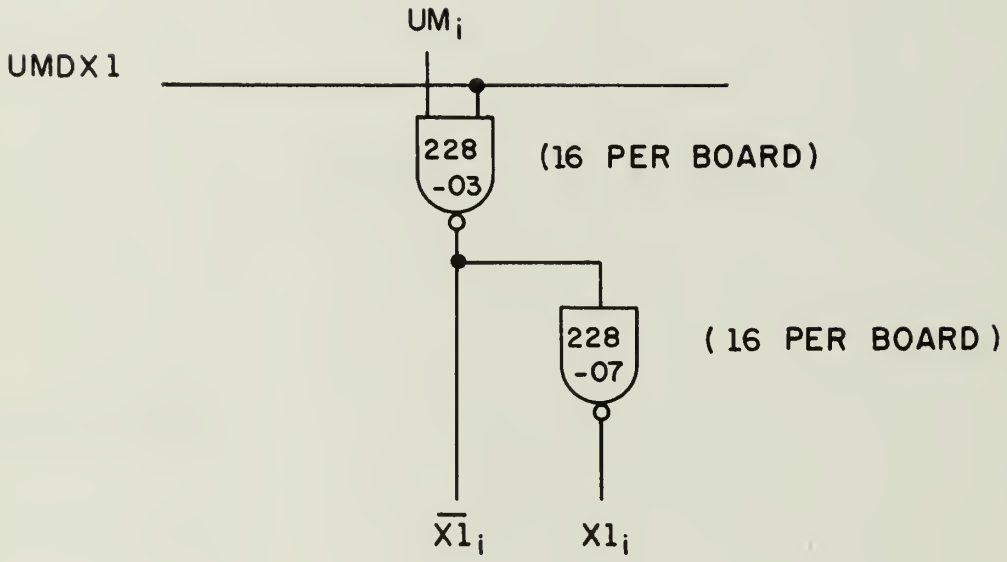


Figure 2.2.3.2.2 - Typical Position of UM Output Gating

```
L/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO UM REGISTER */
ELEVANT DRAWING NUMBERS: 223-01,-02,-03,-04,-05,-06,-07 */
        DECLARE UM BIT(64);/*TRUE OUTPUT OF UM-REGISTER*/
        DECLARE UMSEL BIT(64);/*TRUE OUTPUT OF UM-SELECTOR*/
        DECLARE X1 BIT(64);/*OUTPUT OF UM OUTPUT GATING AND
                             ONE INPUT TO SDS-1*/
M:      /*LOAD UM, ALL BYTES*/
        PROCEDURE;
        CALL LDUM01; CALL LDUM23; CALL LDUM23;
        CALL LDUM45; CALL LDUM67;
        END;
M01:    /*LOAD UM BYTES 0,1*/
        PROCEDURE;
        SUBSTR(UM,1,16)=SUBSTR(UMSEL,1,16);
        END;
M23:    /*LOAD UM BYTES 2,3*/
        PROCEDURE;
        SUBSTR(UM,17,16)=SUBSTR(UMSEL,17,16);
        END;
M45:    /*LOAD UM BYTES 4,5*/
        PROCEDURE;
        SUBSTR(UM,33,16)=SUBSTR(UMSEL,33,16);
        END;
M67:    /*LOAD UM BYTES 6,7*/
        PROCEDURE;
        SUBSTR(UM,49,16)=SUBSTR(UMSEL,49,16);
        END;
UM:     /*SELECT LM DIRECT TO UM, ALL BYTES*/
        PROCEDURE;
        CALL LMDUM03; CALL LMDUM47;
        END;
UM03:   /*SELECT LM DIRECT TO UM BYTES 0-3*/
        PROCEDURE;
        SUBSTR(UMSEL,1,32)=SUBSTR(LM,1,32);
        END;
JM47:   /*SELECT LM DIRECT TO UM, BYTES 4-7*/
        PROCEDURE;
        SUBSTR(UMSEL,33,32)=SUBSTR(LM,33,32);
        END;
BUM:    /*SELECT LM LEFT 8 BITS TO UM, ALL BYTES*/
        PROCEDURE;
        CALL LML8UM03; CALL LML8UM47;
        END;
LUM03:  /*SELECT LM LEFT 8 BITS TO UM, BYTES 0-3*/
        PROCEDURE;
        SUBSTR(UMSEL,1,32)=SUBSTR(LM,9,32);
        END;
LUM47:  /*SELECT LM LEFT 8 BITS TO UM, BYTES 4-7*/
        PROCEDURE;
        SUBSTR(UMSEL,33,32)= SUBSTR(LM,41,24)//(8)'0'B;
        END;
RUM:    /*SELECT LM RIGHT 8 BITS TO UM, ALL BYTES*/
        PROCEDURE;
        CALL LMR8UM03; CALL LMR8UM47;
        END;
RUM03:  /*SELECT LM RIGHT 8 BITS TO UM,BYTES 0-3*/
        PROCEDURE;
```

```
            SUBSTR(UMSEL,1,32)=(8)'0'B//SUBSTR(LM,1,24);
            END
LMR8UM47:   /*SELECT LM RIGHT 8 BITS TO UM,BYTES 4-7*/
            PROCEDURE;
            SUBSTR(UMSEL,33,32)=SUBSTR(LM,25,32);
            END;
UMDX1:      /*SELECT UM DIRECT TO X1, ALL BYTES*/
            PROCEDURE;
            CALL UMDX103; CALL UMDX147;
            END;
UMDX103:    /*SELECT UM DIRECT TO X1, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(X1,1,32)=SUBSTR(UM,1,32);
            END;
UMDX147:    /*SELECT UM DIRECT TO X1, BYTES 4-7*/
            PROCEDURE;
            SUBSTR(X1,33,32)=SUBSTR(UM,33,32);
            END;
UQ912R52UM:/*SELECT UQ, BITS 9-12, RIGHT 52 BITS TO UM (BITS 61-64*/
            PROCEDURE; /* NOTE: ALSO CALLED UQ912DUM6164*/
            SUBSTR(UMSEL,61,4)=SUBSTR(UQ,9,4);
            END;
UQDUM:      /*SELECT UQ DIRECT TO UM, ALL BYTES*/
            PROCEDURE;
            CALL UQDUM02; CALL UQDUM35; CALL UQDUM67;
            END;
UQDUM02:    /*SELECT UQ DIRECT TO UM, BYTES 0-2*/
            PROCEDURE;
            SUBSTR(UMSEL,1,24)=SUBSTR(UQ,1,24);
            END;
UQDUM35:    /*SELECT UQ DIRECT TO UM, BYTES 3-5*/
            PROCEDURE;
            SUBSTR(UMSEL,25,24)=SUBSTR(UQ,25,24);
            END;
UQDUM67:    /*SELECT UQ DIRECT TO UM, BYTES 6,7*/
            PROCEDURE;
            SUBSTR(UMSEL,49,16)=SUBSTR(UQ,49,16);
            END;
```

2.2.4 LS-Register

2.2.4.1 General

The LS-Register (Lower Sign Register) is part of the secondary rank of the accumulator for the signed-digit subtracter (SDS) array. It holds the sign bits of the result from the SDS output prior to their being transferred back to the primary sign bit accumulator (US-Register) or the UH-Register for terminal processing.

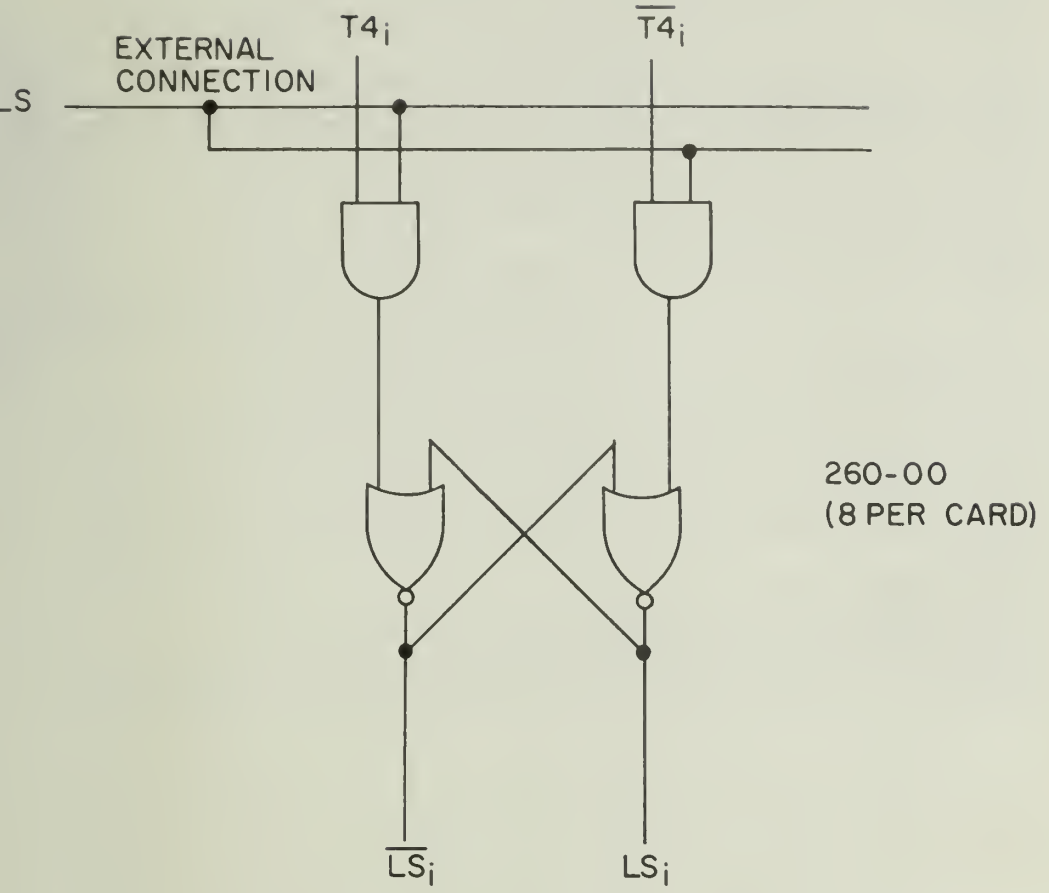
The LS-Register consists of 8 bytes of flip-flop storage. Since this register is loaded from only one source (the T output of subtracter 4) no selector gates such as used with the US-Register are necessary. The input is selected and loaded under control of the signal T4DLS.

The output of the LS-Register drives inputs into the US-selector (LSL8US, LSDUS, LSR8US) and an input into the UH Selector (LSDUH).

2.2.4.2 Implementation

The LS-Register is implemented with the 260-00 flip-flop board. A typical position is shown in Figure 2.2.4.2.1.

Relevant detailed drawings are as follows: 224-01, -02, -03, -04, -05, -06.



TO: FIG. 2.2.2.2.1 (3 LOADS)
FIG. 2.2.6.2.1 (1 LOAD)

Figure 2.2.4.2.1 - Typical Position of the LS Register

2.2.4.3 PL/1 Description of Signal Names

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO LS REGISTER*/
/*RELEVANT DRAWING NUMBERS: 224-01,-02,-03,-04,-05,-06.*/
          DECLARE LS BIT(64); /*TRUE OUTPUT OF LS REGISTER*/
          DECLARE T4 BIT(64); /*TRUE OUTPUT OF SDS-4, SIGN
                                BITS*/
T4DLS:    /*T4 DIRECT TO LS (SAME AS LOAD LS), ALL BYTES*/
          PROCEDURE;
          CALL T4DLS01; CALL T4DLS23; CALL T4DLS45; CALL T4DLS67;
          END;
T4DLS01:  /*T4 DIRECT TO LS, BYTES 0,1 */
          PROCEDURE;
          SUBSTR(LS,1,16)=SUBSTR(T4,1,16);
          END;
T4DLS23:  /*T4 DIRECT TO LS, BYTES 2,3 */
          PROCEDURE;
          SUBSTR(LS,17,16)=SUBSTR(T4,17,16);
          END;
T4DLS45:  /*T4 DIRECT TO LS, BYTES 4,5 */
          PROCEDURE;
          SUBSTR(LS,33,16)=SUBSTR(T4,33,16);
          END;
T4DLS67:  /*T4 DIRECT TO LS, BYTES 6,7 */
          PROCEDURE;
          SUBSTR(LS,49,16)=SUBSTR(T4,33,16);
          END;
```

2.2.5 LM-Register

2.2.5.1 General

The LM-Register (Lower Magnitude Register) is part of the secondary rank of the accumulator for the signed digit subtracter (SDS) array. The register holds the magnitude bits of the result from the SDS output prior to their being transferred back to the primary magnitude bit accumulator (UM-Register) or to the UQ Register which serves as an output buffer.

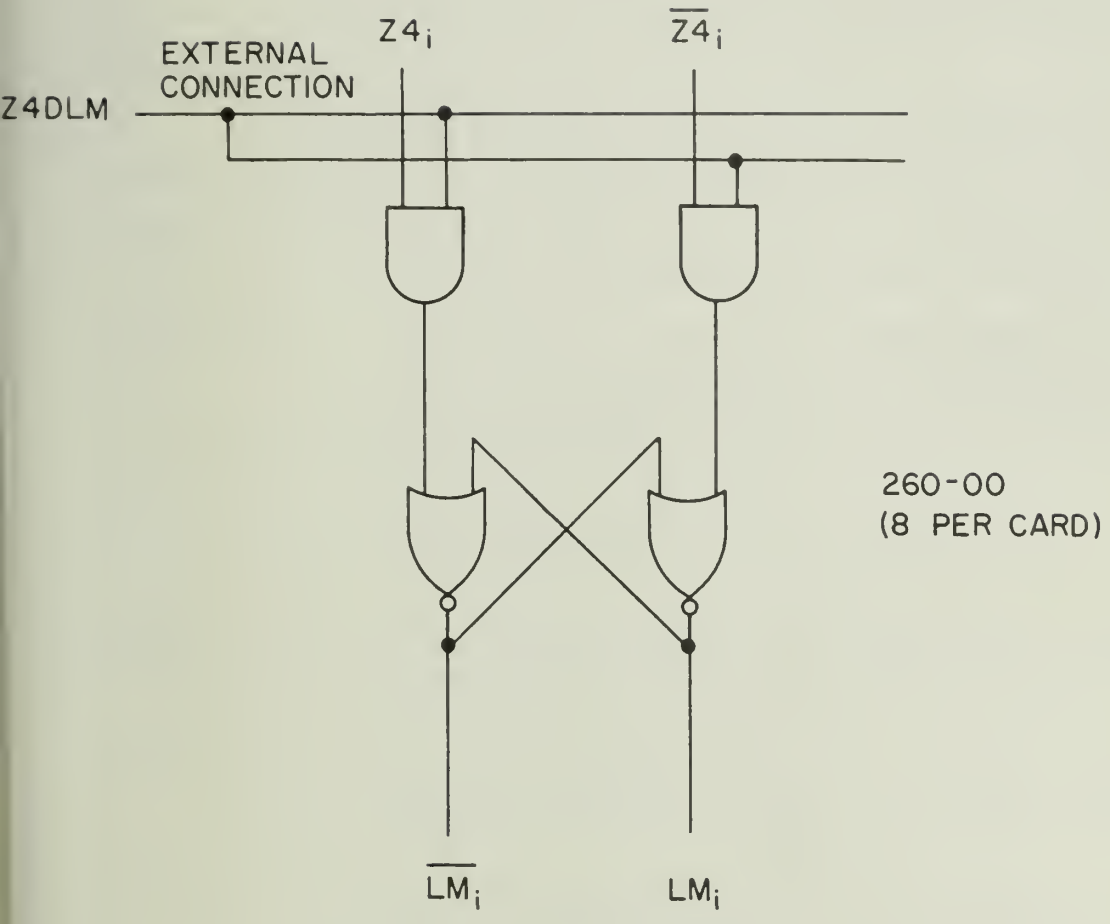
The LM-Register consists of 8 bytes of flip-flop storage. Since this register is loaded from only one source (the Z output of subtractor 4) no selector gates such as used with the UM-Register are necessary. The input is selected and loaded under control of the signal Z4DLS.

The output of the LM-Register drives inputs into the UM-Selector (LML8UM, LMDUM, LMR8UM) and an input into the UQ-Selector (LMDUQ).

2.2.5.2 Implementation

The LM-Register is implemented with the 260-00 flip-flop board. A typical position is shown in Figure 2.2.5.2.1.

Relevant detailed drawings are as follows: 225-01, -02, -03, -04, -05, -06.



TO: FIG. 2.2.3.2.1 (3 LOADS)
FIG. 2.2.7.2.1 (1 LOAD)

Figure 2.2.5.2.1 - Typical Position of the LM Register

2.2.5.3 PL/1 Description of Signal Names

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO LM REGISTER */
/*RELEVANT DRAWING NUMBERS: 225-01,-02,-03,-04,-05,-06 */
          DECLARE LM BIT(64); /*TRUE OUTPUT OF LM REGISTER */
          DECLARE Z4 BIT(64); /*TRUE OUTPUT OF SDS-4, MAGNITUDE
                                BITS*/
Z4DLM:    /*Z4 DIRECT TO LM (SAME AS LOAD LM), ALL BYTES*/
          PROCEDURE;
          CALL Z4SLM01D CALL Z4DLM23; CALL Z4DLM45; CALL Z4DLM67;
          END;
Z4DLM01:  /*Z4 DIRECT TO LM, BYTES 0,1*/
          PROCEDURE;
          SUBSTR(LM,1,16)=SUBSTR(Z4,1,16);
          END;
Z4DLM23:  /*Z4 DIRECT TO LM, BYTES 2,3*/
          PROCEDURE;
          SUBSTR(LM,17,16)=SUBSTR(Z4,17,16);
          END;
Z4DLM45:  /*Z4 DIRECT TO LM, BYTES 4,5*/
          PROCEDURE;
          SUBSTR(LM,33,16)=SUBSTR(Z4,33,16);
          END;
Z4DLM67:  /*Z4 DIRECT TO LM, BYTES 6,7*/
          PROCEDURE;
          SUBSTR(LM,49,16)=SUBSTR(Z4,49,16);
          END:
```

2.2.6 UH-Register

2.2.6.1 General

The UH-Register (Upper H Register) is part of a functional set of registers consisting of UH and UQ and the secondary ranks LH and LQ. These 8-byte registers serve as input-output buffers, as shift registers for initial and terminal operations of an arithmetic order, and as storage for the sign bits of the quotient (in signed-digit format) during division.

The UH-Register complex consists of input selectors and flip-flop storage. The inputs to the register supplied by the selector and their primary use are described below:

Signal Name	Description	Primary Use
LSDUH	Select true outputs of LS-Register direct (i.e. without shifting).	In conversion from integer to decimal number type.
LHL1UH	Select true outputs of LH-Register shifted left 1 bit.	Binary normalization for division.
LHL4UH	Select true outputs of LH-Register shifted left 4 bits.	Hexadecimal normalization in all floating point operations.
LHL8UH	Select true outputs of LH-Register shifted left 8 bits (1 byte)	Hexadecimal normalization in all floating point operations.
LML32UH	Select true outputs of LM-Register shifted left 32 bits (4 bytes).	In fixed point division.
LHR4UH	Select true outputs of LH-Register shifted right 4 bits.	To align radix points of fractions for floating point ADD and SUB.

Signal Name	Description	Primary Use
LHR8UH	Select true outputs of LH-Register shifted right 8 bits.	To align radix points of fractions for floating point ADD and SUB.
VDUH0	Select V-Bus data byte 1 direct to UH byte 0.	Initial loading of operands.
VDUH13	Select V-Bus data bytes 2-4 direct to UH bytes 1-3.	Initial loading of operands.
VDUH47	Select V-Bus data bytes 1-4 direct to UH bytes 4-7.	Initial loading of operands.
QBDUH7	Select the quotient buffer from model division to UH, byte 7.	All division operations.

The outputs of the selector, $UHSEL_i$ (i = 1 to 64), are loaded into the UH flip-flops under control of LDUH (Load UH).

These signals and their sub-signals are defined more precisely in PL/1 in Section 2.2.6.3.

The outputs of the UH flip-flops drive the LH register, the US Selector (UHDUS), the M Selector (UHDM), and the UH Zero Detect logic.

2.2.6.2 Implementation

The UH Selector is implemented with the 294-00 selector card and the 228-03 NAND card. The flip-flops are 260-00. A typical position of the UH Register and Selector is shown in Figure 2.2.6.2.1.

Relevant detailed drawings are as follows: 226-01, 02, -03, -04, -05, -06, -07.

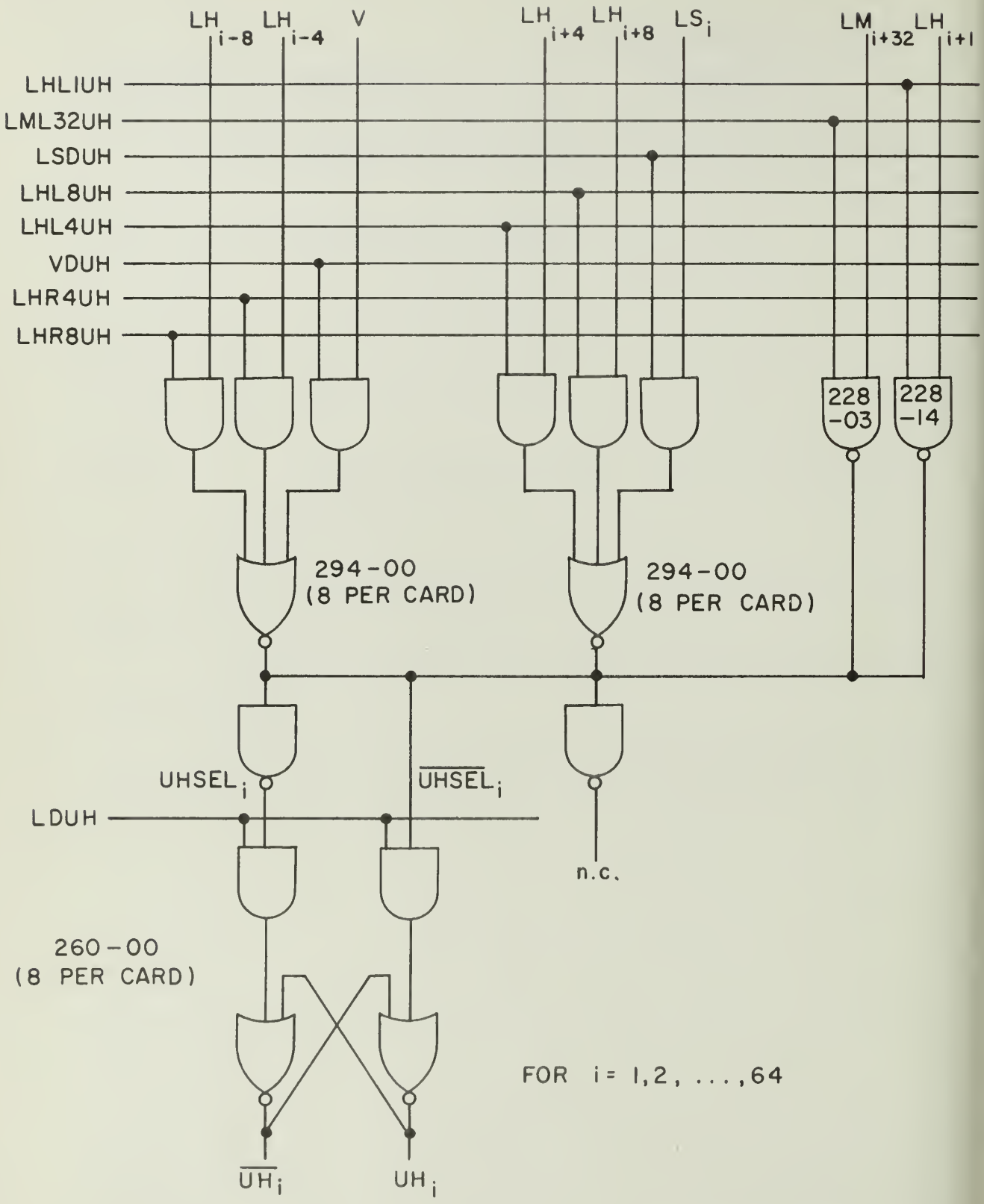


Figure 2.2.6.2.1 - Typical Position of the UH Register and Selectors

2.2.6.3 PL/1 Description of Signal Names

```
L/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO UH REGISTER*/
ELEVANT DRAWING NUMBERS: 22 -01,-02,-03,-04,-05,-06,-07*/
        DECLARE UH BIT(64); /*TRUE OUTPUT OF UH REGISTER*/
        DECLARE UHSEL BIT(64); /*TRUE OUTPUT OF UH SELECTOR*/
H:      PROCEDURE;
        CALL LDUH01; CALL LDUH23;
        CALL LDUH45; CALL LDUH67;
        END;
H03:    PROCEDURE;
        CALL LDUH01; CALL LDUH23;
        END;
H47:    PROCEDURE;
        CALL LDUH45; CALL LDUH67;
        END;
H01     /*LOAD UH BYTES 0,1 FROM UHSEL BYTES 0,1 */
        PROCEDURE;
        SUBSTR(UH,1,16)=SUBSTR(UHSEL,1,16);
        END;
H23:    /*LOAD UH BYTES 2,3 FROM UHSEL BYTES 2,3 */
        PROCEDURE;
        SUBSTR(UH,17,16)=SUBSTR(UHSEL,17,16);
        END;
H45:    /*LOAD UH BYTES 4,5 FROM UHSEL BYTES 4,5 */
        PROCEDURE;
        SUBSTR(UH,33,16)=SUBSTR(UHSEL,33,16);
        END;
H67:    /*LOAD UH BYTES 6,7 FROM UHSEL BYTES 6,7 */
        PROCEDURE;
        SUBSTR(UH,49,16)=SUBSTR(UHSEL,49,16);
        END;
1UH:    /*SELECT  LH LEFT 1 BIT TO UH*/
        PROCEDURE;
        CALL LHL1UH14; CALL LHL1UH57;
        END;
1UH14:  /*SELECT LH LEFT 1 BIT TO UH BYTES 1-4*/
        PROCEDURE;
        SUBSTR(UHSEL,9,32)=SUBSTR(LH,10,32);
        END;
1UH57:  /*SELECT LH LEFT 1 BIT TO UH BYTES 5-7*/
        PROCEDURE;
        SUBSTR(UHSEL,41,24)=SUBSTR(LH,42,23)||'0'B;
        END;
L4UH:   /*SELECT LH LEFT 4 BITS TO UH, ALL BYTES*/
        PROCEDURE;
        CALL LHL4UH03; CALL LHL4UH47;
        END;
L4UH03: /*SELECT LH LEFT 4 BITS TO UH, BYTES 0-3*/
        PROCEDURE;
        SUBSTR(UHSEL,1,32)=SUBSTR(LH,5,32);
        END;
L4UH47: /*SELECT LH LEFT 4 BITS TO UH, BYTES 4-7*/
        PROCEDURE;
        SUBSTR(UHSEL,33,32)=SUBSTR(LH,37,28)||'0000'B;
        END;
L8UH:   /*SELECT LH LEFT 8 BITS TO UH, ALL BYTES*/
        PROCEDURE;
        CALL LHL8UH03; CALL LHL8UH47;
```

```
            END;
LHL8UH03:   /*SELECT LH LEFT 8 BITS TO UH, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(UHSEL,1,32)=SUBSTR(LH,9,32);
            END;
LHL8UH47:   /*SELECT LH LEFT 8 BITS TO UH, BYTES 4-7*/
            PROCEDURE;
            SUBSTR(UHSEL,33,32)=SUBSTR(LH,41,24)||8'0'B;
            END;
LHR4UH:     /*SELECT LH RIGHT 4 BITS TO UH, ALL BYTES*/
            PROCEDURE;
            CALL LHR4UH03; CALL LHR4UH47;
            END;
LHR4UH03:   /*SELECT LH RIGHT 4 BITS TO OH, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(UHSEL,1,32)='0000'B||SUBSTR(LH,1,28);
            END;
LHR4UH47:   /*SELECT LH RIGHT 4 BITS TO UH, BYTES 4-7*/;
            PROCEDURE;
            SUBSTR(UHSEL,33,32)=SUBSTR(LH,29,32);
            END;
LHR8UH:     /*SELECT LH RIGHT 8 BITS TO UH, ALL BYTES*/
            PROCEDURE;
            CALL LHR8UH03; CALL LHR8UH47;
            END;
LHR8UH03:   /*SELECT LH RIGHT 8 BITS TO UH, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(UHSEL,1,32)=(8)'0'B||SUBSTR(LH,1,24);
            END;
LHR8UH47:   /*SELECT LH RIGHT 8 BITS TO UH, BYTES 4-7*/
            SUBSTR(UHSEL,33,32)=SUBSTR(LH,25,32);
            END;
LML32UH:    /*SELECT LM LEFT 32 BITS INTO UH (SAME AS LMDUH7)*/
            PROCEDURE;
            SUBSTR(UHSEL,1,32)=SUBSTR(LM,33,32);
            END;
LSDUH:      /*SELECT LS DIRECT TO UH, ALL BYTES*/
            PROCEDURE;
            CALL LSDUH03; CALL LSDUH47;
            END;
LSDUH03:    /*SELECT LS DIRECT TO UH, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(UHSEL,1,32)=SUBSTR(LS,1,32);
            END;
LSDUH47:    /*SELECT LS DIRECT TO UH, BYTES 4-7*/
            PROCEDURE;
            SUBSTR(UHSEL,33,32)=SUBSTR(LS,33,32);
            END;
VDUH0:      /*SELECT V-BUS DATA BYTE 1 DIRECT TO UH BYTE 0 */
            PROCEDURE;
            SUBSTR(UHSEL,1,8)=SUBSTR(V,11,8);
            END;
VDUH13:     /*SELECT V-BUS DATA BYTES 2-4 DIRECT TO UH BYTES 1-3*/
            PROCEDURE;
            SUBSTR(UHSEL,9,8)=SUBSTR(V,21,8);
            SUBSTR(UHSEL,17,8)=SUBSTR(V,31,8);
            SUBSTR(UHSEL,25,8)=SUBSTR(V,41,8);
```

```
        END;
UH47:   /*SELECT V-BUS DATA BYTES 1-4 DIRECT TO UH BYTES 4-7*/
        PROCEDURE;
        SUBSTR(UHSEL,33,8)=SUBSTR(V,11,8);
        SUBSTR(UHSEL,41,8)=SUBSTR(V,21,8);
        SUBSTR(UHSEL,49,8)=SUBSTR(V,31,8);
        SUBSTR(UHSEL,57,8)=SUBSTR(V,41,8);
        END;
```

2.2.7 UQ-Register

2.2.7.1 General

The UQ-Register (Upper Q Register) is part of a functional set of registers consisting of UH and UQ and the secondary ranks LH and LQ, respectively. This 8-byte register serves as an input-output buffer, as a shift register for initial and terminal operations of an arithmetic order and as storage for the magnitude bits of the quotient (in signed-digit format) during division.

The UQ-Register complex consists of input selectors and flip-flop storage. The inputs to the register and their primary use are described below:

Signal Name	Description	Primary Use
LMDUQ	Select true output of LM-Register directly (without shifting)	Transfer of results for signe digit subtracter complex to U for terminal shifting and out
LQL1UQ	Select true output of LQ-Register shifted left 1 bit.	Binary normalization for divi
LQL4UQ	Select true output of LQ-Register shifted left 4 bits.	Hexadecimal normalization in floating point operations.
LQL8UQ	Select true output of LQ-Register shifted left 8 bits.	Hexadecimal normalization in floating point operations.
LQR1UQ03	Select true output of LQ-Register shifted right 1 bit. Applies only to bytes 0-3.	Post normalization of remaind in integer division.
LQR4UQ	Select true output of LQ-Register shifted right 4 bits.	Alignment of radix points of fractions for floating point ADD and SUB.
LQR8UQ	Select true output of LQ-Register shifted right 8 bits.	Alignment of radix point of fractions for floating point ADD and SUB.

Signal Name	Description	Primary Use
SELDECSIGN	Select the sign of a decimal result into UQ(5) - UQ(8)	Conversion to decimal number type.
SETUQ030NE	Set UQ(1) - UQ(32) to all 1's.	Generation of maximum floating point number on division by zero.
VDUQ0	Select V-Bus data byte 1 direct to UQ byte 0.	Initial loading of operands.
VDUQ13	Select V-Bus data bytes 2-4 direct to UQ bytes 1-3.	Initial loading of operands.
VDUQ47	Select V-Bus data bytes 1-4 direct to UQ bytes 4-7.	Initial loading of operands.

The outputs of the UQ selector, UQSELi (i = 1 to 64) are loaded into the UQ flip-flops under control of LDUQ (Load UQ).

All signals associated with the UQ-Register are defined more precisely in PL/1 notation in Section 2.2.7.3.

The outputs of the UQ flip-flops drive the LQ register, the UM selector, the UQ Zero Detect Logic and the XT-Bus interface.

2.2.7.2 Implementation

The UQ Selector is implemented with the 294-00 selector card and the 228-03 NAND card. The flip-flops are the 260-00. A typical position of the high order half of the UH Register and Selector is shown in Figure 2.2.7.2.1. A typical position of the low-order half is shown in Figure 2.2.7.2.2.

Relevant detailed drawings are as follows: 227-01, 02, 03, 04, 05, 06, 07.

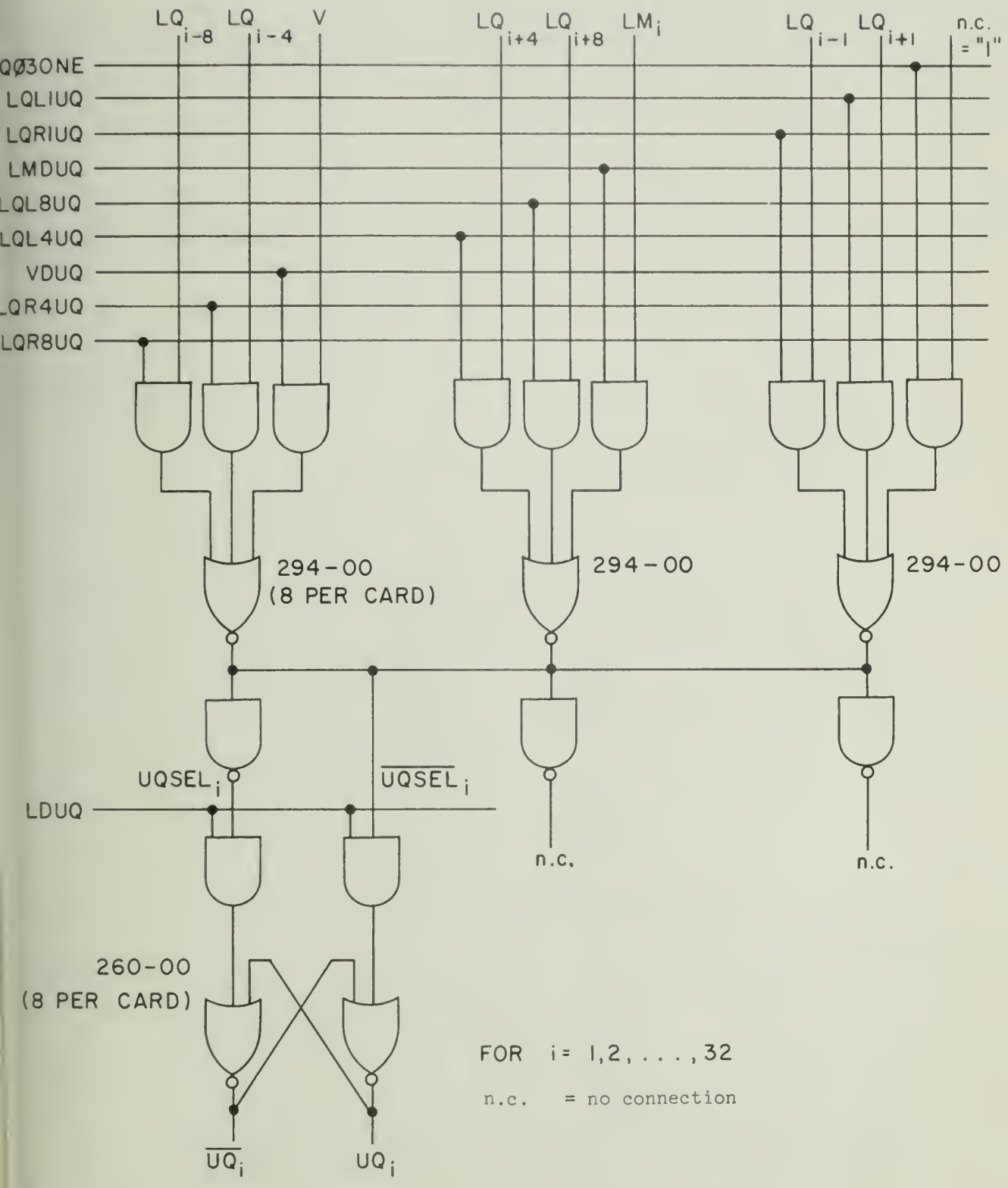


Figure 2.2.7.2.1 - Typical Position of UQ Register and Selector (Bytes 0-3)

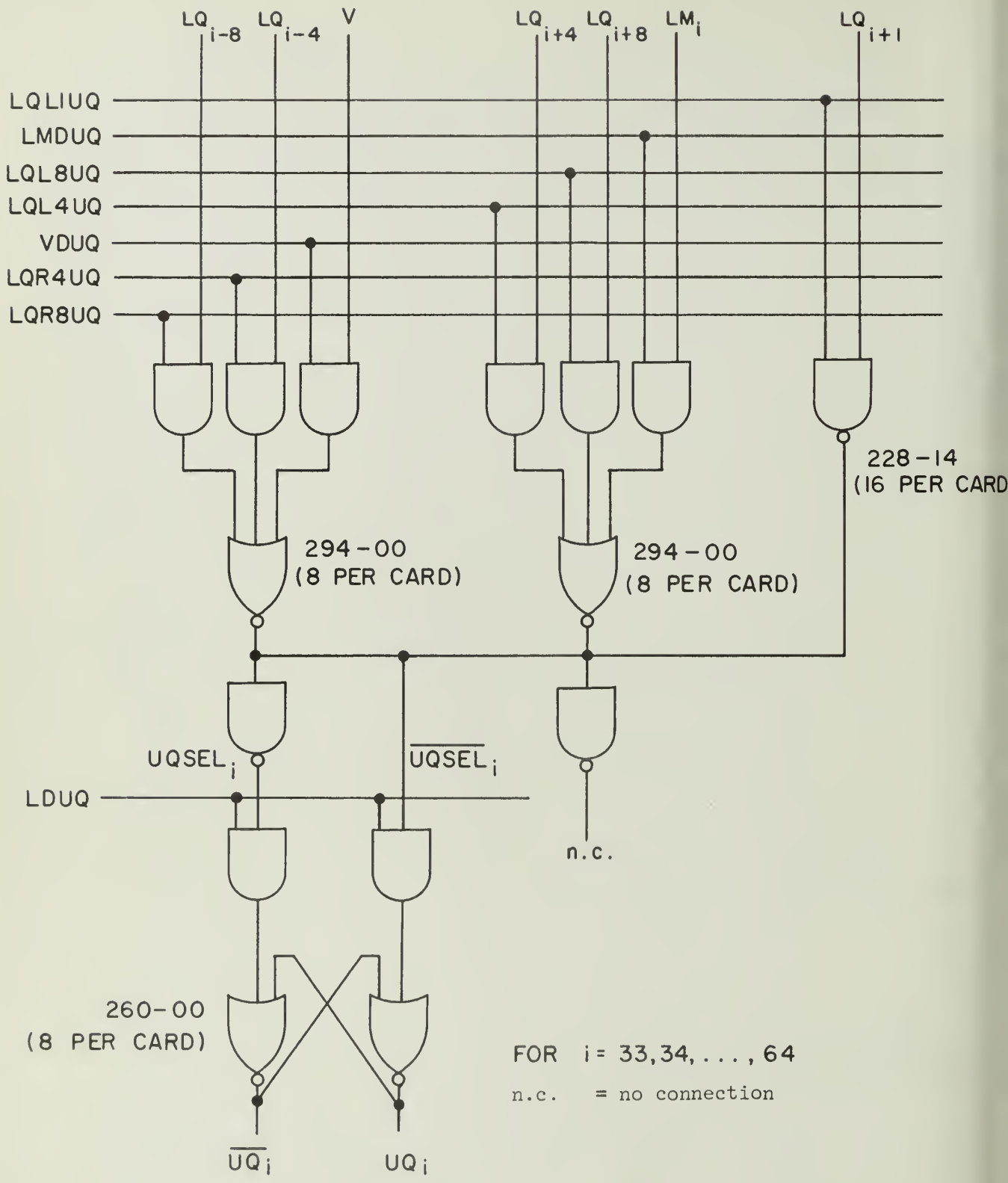


Figure 2.2.7.2.2 - Typical Position of UQ Register and Selector (Bytes 4-7)

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO UQ REGISTER*/
/*RELEVANT DRAWING NUMBERS: 227-01,-02,-03,-04,-05,-06,-07*/
          DECLARE UQ BIT(64); /*TRUE OUTPUT OF UQ REGISTER*/
          DECLARE UQSEL BIT(64); /*TRUE OUTPUT OF UQ SELECTOR*/
          DECLARE  TEMP_S BIT(4), TEMP1_S BIT(8), INJECTSIGN_S
                   BIT(4), INJECTSIGN1_S BIT(8);
INJECTSIGN:/*INJECT SIGN IN HIGH ORDER POSITIONS OF UQ ON RIGHT
            SHIFTS */
          PROCEDURE;
          INJECTSIGN_S = (4)'1'B; INJECTSIGN1_S = (8)'1'B;
          END;
LDUQ:     PROCEDURE;
          CALL LDUQ0; CALL LDUQ1; CALL LDUQ23;
          CALL LDUQ45; CALL LDUQ67;
          END;
LDUQ13:   PROCEDURE;
          CALL LDUQ1; CALL LDUQ23;
          END;
LDUQ47:   PROCEDURE;
          CALL LDUQ45; CALL LDUQ67;
          END;
LDUQ0:    /*LOAD UQ BYTE 0 FROM UQSEL BYTE 0 */
          PROCEDURE;
          SUBSTR(UQ,1,8)=SUBSTR(UQSEL,1,8);
          END;
LDUQ1:    /*LOAD UQ BYTE 1 FROM UQSEL BYTE 1 */
          PROCEDURE;
          SUBSTR(UQ,9,8)=SUBSTR(UQSEL,9,8);
          END;
LDUQ23:   /*LOAD UQ BYTES 2,3 FROM UQSEL BYTES 2,3 */
          PROCEDURE;
          SUBSTR(UQ,17,16)=SUBSTR(UQSEL,17,16);
          END;
LDUQ45:   /*LOAD UQ BYTES 4,5 FROM UQSEL BYTES 4,5 */
          PROCEDURE;
          SUBSTR(UQ,33,16)=SUBSTR(UQSEL,33,16);
          END;
LDUQ67:   /*LOAD UQ BYTES 6,7 FROM UQSEL BYTES 6,7 */
          PROCEDURE;
          SUBSTR(UQ,49,16)=SUBSTR(UQSEL,49,16);
          END;
LMDUQ:    /*SELECT LM DIRECT TO UQ, ALL BYTES*/
          PROCEDURE;
          CALL LMDUQ03; CALL LMDUQ47;
          END;
LMDUQ03:  /*SELECT LM DIRECT TO UQ, BYTES 0-3*/
          PROCEDURE;
          SUBSTR(UQSEL,1,32)=SUBSTR(LM,1,32);
          END;
LMDUQ47:  /*SELECT LM DIRECT TO UQ, BYTES 4-7*/
          PROCEDURE;
          SUBSTR(UQSEL,33,32)=SUBSTR(LM,33,32);
          END;
LQL1UQ:   /*SELECT LQ LEFT 1 BIT TO UH*/
          PROCEDURE;
          CALL LQL1UQ04; CALL LQL1UQ57;
          END;
```

```
LQL1UQ04:   /*SELECT LQ LEFT 1 BIT TO UQ, BYTES 0-4*/
            PROCEDURE;
            SUBSTR(UQSEL,1,40)=SUBSTR(LQ,2,40);
            END;
LQL1UQ57:   /*SELECT LQ LEFT 1 BIT TO UQ, BYTES 5-7*/
            PROCEDURE;
            SUBSTR(UQSEL,41,24)=SUBSTR(LQ,42,23)||'0'B;
            END;
LQL4UQ:     /*SELECT LQ LEFT 4 BITS TO UQ, ALL BYTES*/
            PROCEDURE;
            CALL LQL4UQ03; CALL LQL4UQ47;
            END;
LQL4UQ03:   /*SELECT LQ LEFT 4 BITS TO UQ, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(UQSEL,1,32)=SUBSTR(LQ,5,32);
            END;
LQL4UQ47:   /*SELECT LQ LEFT 4 BITS TO UQ, BYTES 4-7*/
            PROCEDURE;
            SUBSTR(UQSEL,33,32)=SUBSTR(LQ,37,28)||MEPB;
            END:
LQL8UQ:     /*SELECT LQ LEFT 8 BITS TO UQ, ALL BYTES*/
            PROCEDURE;
            CALL LQL8UQ03; CALL LQL8UQ47;
            END;
LQL8UQ03:   /*SELECT LQ LEFT 8 BITS TO UQ, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(UQSEL,1,32)=SUBSTR(LQ,9,32);
            END;
LQL8UQ47:   /*SELECT LQ LEFT 8 BITS TO UQ, BYTES 4-7*/
            PROCEDURE;
            SUBSTR(UQSEL,33,32)=SUBSTR(LQ,41,24)||MEPB||'000'B;
            END;
LQR1UQ03:   /*SELECT LQ RIGHT 1 BIT TO UQ, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(UQSEL,1,32)=SUBSTR(LQ,1,1)||SUBSTR(LQ,1,31);
            /*NOTE THAT SIGN IS INJECTED IN HIGH ORDER POSITION*/
            END;
LQR4UQ:     /*SELECT LQ RIGHT 4 BITS TO UQ, ALL BYTES*/
            PROCEDURE,
            CALL LQR4UQ03; CALL LQR4UQ47;
            END;
LQR4UQ03:   /*SELECT LQ RIGHT 4 BITS TO UQ, BYTES 0-3*/
            PROCEDURE; TEMP_S =INJECTSIGN_S & (4)SUBSTR(LQ,1,1);
            SUBSTR(UQSEL,1,32)= TEMPS||SUBSTR(UQ,1,28);
            END;
LQR4UQ47:   /*SELECT LQ RIGHT 4 BITS TO UQ, BYTES 4-7*/;
            PROCEDURE;
            SUBSTR(UQSEL,33,32)=SUBSTR(LQ,29,32);
            END;
LQR8UQ:     /*SELECT LQ RIGHT 8 BITS TO UQ, ALL BYTES*/
            PROCEDURE;
            CALL LQR8UQ03; CALL LQR8UQ47;
            END;
LQR8UQ03:   /*SELECT LQ RIGHT 8 BITS TO UQ, BYTES 0-3*/
            PROCEDURE; TEMP1_S=INJECTSIGN1_S & (8)SUBSTR(LQ,1,1);
            SUBSTR(UQSEL,1,32)=TEMP1_S||SUBSTR(LQ,1,24);
            END;
```

```
LQR8UQ47:   /*SELECT LQ RIGHT 8 BITS TO UQ, BYTES 4-7*/
            SUBSTR(UQSEL,33,32)=SUBSTR(LQ,25,32);
            END;
SETDECSIGN:/*SET DECIMAL SIGN IN UQ BITS 5-8*/
            PROCEDURE;
            SUBSTR(UQSEL,5,4)='101'||SR; /* SR=SIGN OF RESULT*/
            END;
SETUQ03ONE:/*SET UQ BYTES 0-3 TO ALL  '1'B */
            PROCEDURE;
            SUBSTR(UQSEL,1,32)=(32)'1'B;
            END;
VDUQ0:      /*SELECT V-BUS DATA BYTE 1 DIRECT TO U  BYTE 0 */
            PROCEDURE;
            SUBSTR(UQSEL,1,8)=SUBSTR(V,11,8);
            END;
VDUQ13:     /*SELECT V-BUS DATA BYTES 2-4 DIRECT TO UQ BYTES 1-3*/
            PROCEDURE;
            SUBSTR(UQSEL,9,8)=SUBSTR(V,21,8);
            SUBSTR(UQSEL,17,8)=SUBSTR(V,31,8);
            SUBSTR(UQSEL,25,8)=SUBSTR(V,41,8);
            END;
VDUQ47:     /*SELECT V-BUS DATA BYTES 1-4 DIRECT TO UQ BYTES 4-7*/
            PROCEDURE;
            SUBSTR(UQSEL,33,8)=SUBSTR(V,11,8);
            SUBSTR(UQSEL,41,8)=SUBSTR(V,21,8);
            SUBSTR(UQSEL,49,8)=SUBSTR(V,31,8);
            SUBSTR(UQSEL,57,8)=SUBSTR(V,41,8);
            END:
```

2.2.8 LH-Register

2.2.8.1 General

The LH-Register (Lower H Register) is the secondary rank associated with the UH-Register. The LH-Register consists of 8 bytes (64 bits) of flip-flop storage. Since this register is loaded from only one source (the true outputs of the UH-Register) no selector gates are required. The input is selected and loaded under control of the signal UHDLH. The outputs of the LH-Register drive the inputs to the US-Selector (LHL8UH , LHL4UH, LHR8UH, LHR4UH, LHL1UH).

2.2.8.2 Implementation

The LH-Register is implemented with the 260-00 flip-flop board. A typical position is shown in Figure 2.2.8.2.1.

Relevant detailed drawings are as follows: 228-01, -02, -03, -04, -05, -06.

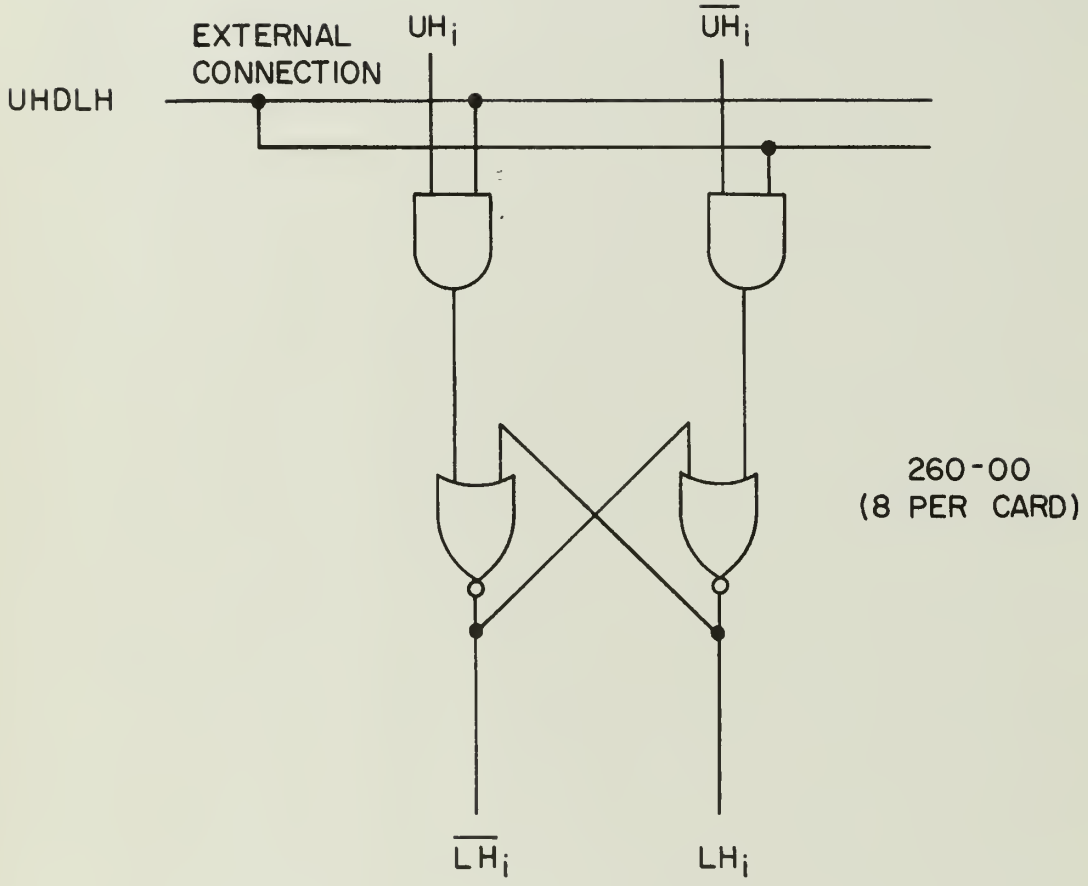


Figure 2.2.8.2.1 - Typical Position of the LH-Register

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO LH REGISTER*/
/*RELEVANT DRAWING NUMBERS: 228-01,-02,-03,-04,-05,-06.*/
          DECLARE LH BIT(64); /*TRUE OUTPUT OF LH REGISTER*/
UHDLH:    /*UH DIRECT TO LH (SAME AS LOAD LH), ALL BYTES*/
          PROCEDURE;
          CALL UHDLH01; CALL UHDLH23; CALL UHDLH45; CALL UHDLH67;
          END;
UHDLH01:  /*UH DIRECT TO LH, BYTES 0,1 */
          PROCEDURE;
          SUBSTR(LH,1,16)=SUBSTR(UH,1,16);
          END;
UHDLH23:  /*UH DIRECT TO LH, BYTES 2,3 */
          PROCEDURE;
          SUBSTR(LH,17,16)=SUBSTR(UH,17,16);
          END;
UHDLH45:  /*UH DIRECT TO LH, BYTES 4,5 */
          PROCEDURE;
          SUBSTR(LH,33,16)=SUBSTR(UH,33,16);
          END;
UHDLH67:  /*UH DIRECT TO LH, BYTES 6,7 */
          PROCEDURE;
          SUBSTR(LH,49,16)=SUBSTR(UH,33,16);
          END;
```

2.2.9 LQ-Register

2.2.9.1 General

The LQ-Register (Lower Q-Register) is the secondary rank associated with the UQ-Register. The LQ-Register consists of 8 bytes (64 bits) of flip-flop storage. Since this register is loaded from only one source (the true outputs of the UQ-Register) no selector gates are required. The input is selected and loaded under control of the signal UQDLQ. The outputs of the LQ-Register drive the inputs to the UQ-Selector(LQL8UH, LQL4UQ, LQR8UQ, LQR4UQ, LQL1UQ, LQR1UQ).

2.2.9.2 Implementation

The LQ-Register is implemented with the 260-00 flip-flop board. A typical position is shown in Figure 2.2.9.2.1. Relevant detailed drawings are as follows: 229-01, -02, -03, -04, -05, -06.

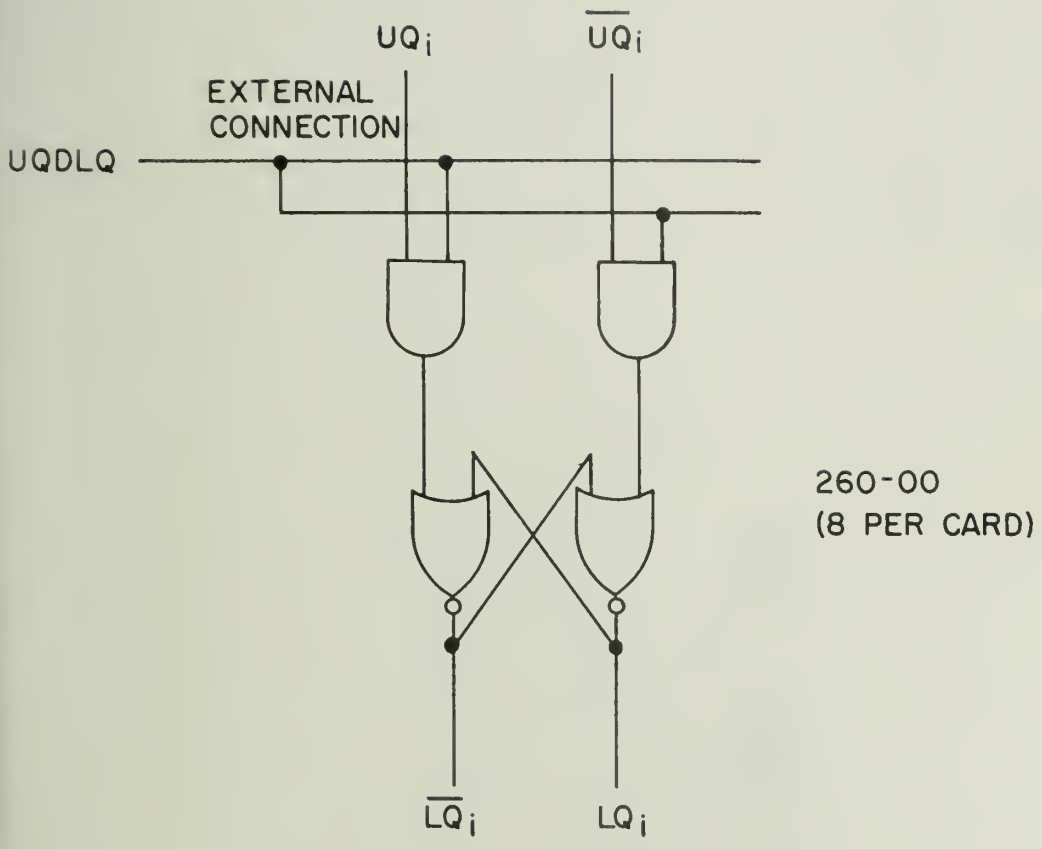


Figure 2.2.9.2.1 - Typical Position of the LQ-Register

2.2.9.3 PL/1 Description of Signal Names

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO LQ REGISTER */
/*RELEVANT DRAWING NUMBERS: 229-01,-02,-03,-04,-05,-06 */
          DECLARE LQ BIT(64); /*TRUE OUTPUT OF LQ REGISTER */
UQDLQ:    /*UQ DIRECT TO LQ (SAME AS LOAD LQ), ALL BYTES*/
          PROCEDURE;
          CALL UQDLQ01; CALL UQDLQ23; CALL UQDLQ45; CALL UQDLQ67;
          END;
UQDLQ01:  /*UQ DIRECT TO LQ, BYTES 0,1*/
          PROCEDURE;
          SUBSTR(LQ,1,16)=SUBSTR(UQ,1,16);
          END;
UQDLQ23:  /*UQ DIRECT TO LQ  BYTES 2,3*/
          PROCEDURE;
          SUBSTR(LQ,17,16)=SUBSTR(UQ,17,16);
          END;
UQDLQ45:  /*UQ DIRECT TO LQ, BYTES 4,5*/
          PROCEDURE;
          SUBSTR(LQ,33,16)=SUBSTR(UQ,33,16);
          END;
UQDLQ67:  /*UQ DIRECT TO LQ, BYTES 6,7*/
          PROCEDURE:
          SUBSTR(LQ,49,16)=SUBSTR(UQ,49,16);
          END;
```

2.3 V-BUS Interface

2.3.1 General

All operands and control signals from a Taxicrinic Processor enter an AU by way of the V-BUS. The 'V' is mnemonic if it is considered to be an arrowhead on a bus entering the AU. The V-BUS interface consists of line terminator boards, inverters, and parity check logic.

The input to the V-BUS interface is driven by the INBUS local exchange number 5 of the Exchange Net. The INBUS consists of five, 10-bit bytes. The bytes are designated 0 through 4. The bits are designated 0 through 49. The bits of the V-BUS (V_i) are designated 1 through 50; V_1 corresponds to INBUS bit 0, etc. The byte numbering is identical for both the INBUS and the V-BUS. Byte 0 is the control byte. The significance of each bit of the control byte is defined in the next section. The remaining bytes each consist of 8 bits of data, 1 flag bit and a parity bit. The parity bit is appended to each byte of data transmitted over the Exchange Net. The parity convention is such that the sum of 1's across all 10 bits should be an odd number.

For transmission from a TP to an AU the parity of each byte of a data word is checked as it is gated into an AU register. If a parity error is detected, an indicator is set in the AU and transmission of the operands continues to completion. The AU, however, does not execute the instruction but rather requests a return path via the Exchange Net to the appropriate TP. Bit number 5 of the AU OUTBUS control byte (XT_6) is set to '1'. The data word sent with the control byte is all zeros. The AU clears and resets to await another request. The TP, having been notified of the parity error, signals an interrupt.

Table 2.3.1.1 summarizes the format of the V-BUS.

INBUS BIT NO.	V-BUS BIT NO.	DESCRIPTION
0-9	1-10	Control byte (See Tables 2.3.2.1 and 2.3.2.2)
10-17	11-18	Data bits of byte 1
18	19	Flag bit of byte 1
19	20	Parity bit of byte 1
20-27	21-28	Data bits of byte 2
28	29	Flag bit of byte 2
29	30	Parity bit of byte 2
30-37	31-38	Data bits of byte 3
38	39	Flag bit of byte 3
39	40	Parity bit of byte 3
40-47	41-48	Data bits of byte 4
48	49	Flag bit of byte 4
49	50	Parity bit of byte 4

Table 2.3.1.1 - Format of INBUS/V-BUS

2.3.2 Input to the Arithmetic Units via the V-BUS

The structure of the control byte which is sent to an Arithmetic Unit when its services are needed is shown in the Table 2.3.2.1. A brief description of each signal names is given in Table 2.3.2.2. The nomenclature is consistent with that defined in File No. 790, "A Discussion of Illiac III Processor-Unit Communication via the Exchange Net". Note that all signals are transmitted through the Exchange Net in the logically complemented form.

Assuming that the reader is now familiar with the signal names, the next section describes the signal sequencing for a TP to AU transmission.

Having recognized the need for an AU, a TP sets PREN = "1" with the Unit Address bits 5-8 of TP-EN Inbus Interface) set to 1111 = 15_{10}. The unit address must be valid before, and 100 nsec. after, PREN = 1. One hundred nsec. after PREN = 1 but before any TII signals are generated, the unit address lines are changed to contain instruction variant and number type information.

Within the Exchange Net, the TP is assigned to an AU and the TP identification number is copied into the assigned AU Identification Register (contained within the EN). The contents of this register are used by the AU in returning results to the TP which requested it.

When the INBUS path to an AU has been secured, the Exchange Net sends a reply, ENRP = 1, back to the requesting TP. Once ENRP = 1 the TP may transfer information to the AU at any time it is available. The TP transfers operands by generating a sequence of TII signals TII0, TII1, TII2, TII3, on the TII line. Associated with each TII signal is a full word of data and one control byte on the INBUS which must be valid during the time each TII signal is valid. The initial design values for the timing relationship between valid data and the TII signals are given in Figure 2.3.2.4.

TP - EN Interface	Exchange Net INBUS Bit No.	V-BUS Bit No.	AU - EN Interface
$\overline{\text{PREN}}$	0	1	$\overline{\text{PREN}}$
(none)	* 1 *	2	$\overline{\text{ENRU}}$
$\overline{\text{TII}}$ (Transfer Information In)	2	3	$\overline{\text{TII}}$
$\overline{\text{IV0}} = \overline{\text{UC0}}$	3	4	$\overline{\text{IV0}}$
$\overline{\text{IV1}} = \overline{\text{UC1}}$	4	5	$\overline{\text{IV1}}$
$\overline{\text{IV2}} = \overline{\text{UC2}}$	5	6	$\overline{\text{IV2}}$
$\overline{\text{IV3}} = \overline{\text{UC3}}$	6	7	$\overline{\text{IV3}}$
$\overline{\text{NT0}} = \overline{\text{UC4}}$	7	8	$\overline{\text{NT0}}$
$\overline{\text{NT1}} = \overline{\text{UC5}}$	8	9	$\overline{\text{NT1}}$
(none)	* 9 *	10	$\overline{\text{IAU0}}$

*Indicates that this line does not go through the Exchange Net.

Table 2.3.2.1 - INBUS Control Bit Assignment

Name	Description	Equivalent Name in AU Documentation if Different
PREN *	Processor Requests Exchange Net	AU Request
ENRU *	Exchange Net Reply to Unit	---
TII *	Transfer Information In	In Info Ready
IV_n	Instruction Variant, Bit n	---
NT_n	Number Type, Bit n	---
UC_n *	Unit Command, Bit n (These bits are assigned to instruction variant and number type bits as shown in Table 1.5.2.	---
IAU0	Interrupt AU 0	Interrupt

*Standard Signals used in control bytes for all Processors and Units. See DCS File No. 790.

Table 2.3.2.2 - Description of INBUS Control Signals

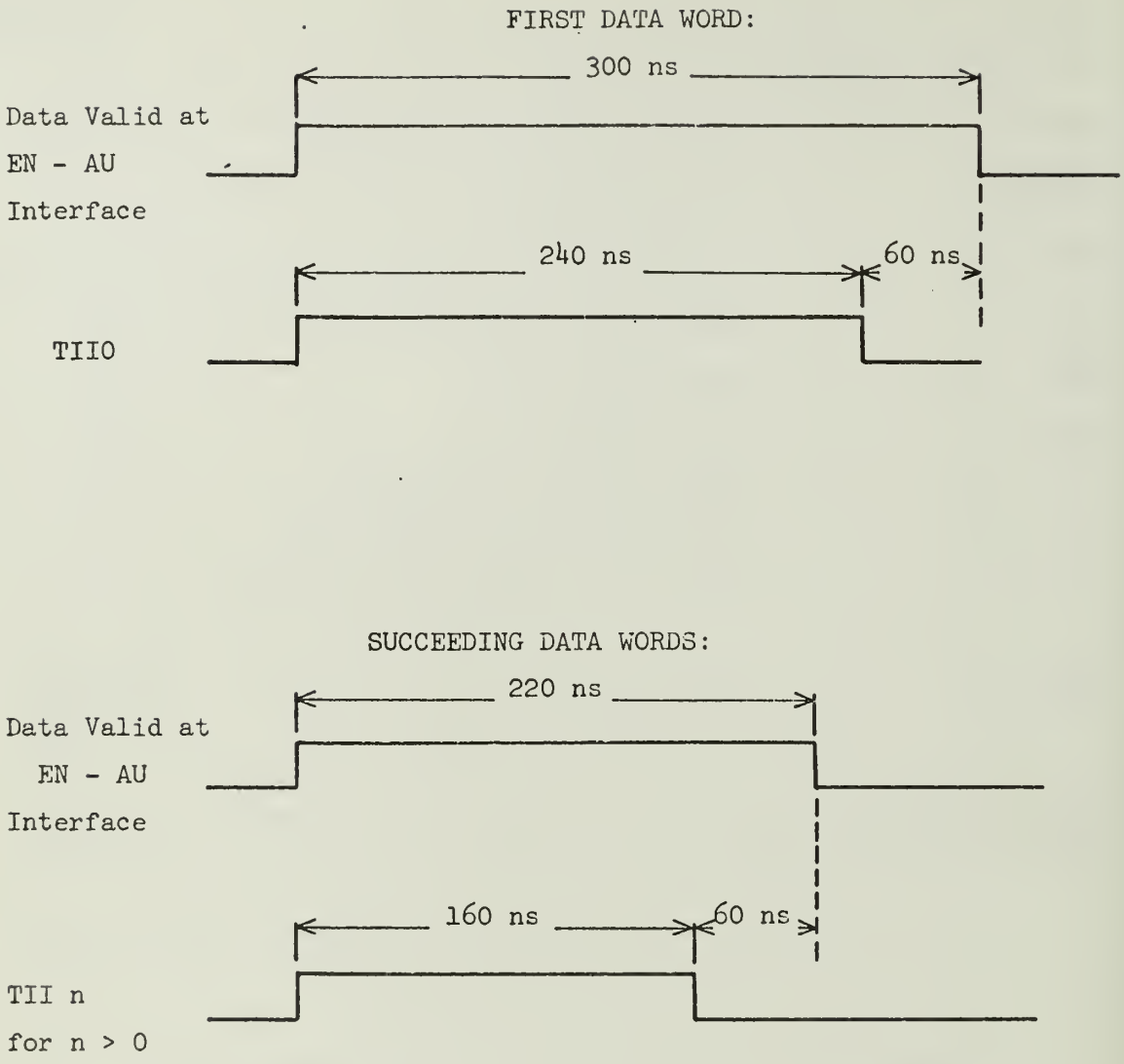


Figure 2.3.2.1 - Relative Timing Between Valid Data on INBUS and TII

Data must be valid longer for the first transmission since the instruction variant and number type must be decoded prior to the loading of the data (the IV and NT codes are given in Figure 2.3.2.3. This initial transmission case is illustrated at the top of Figure 2.3.2.4. The timing for all successive transmissions is shown at the bottom of this figure. In both cases, the data must remain valid at least 60 nsec. after the TII is turned off. The minimum interval required between successive TII pulses is only about 20 nsec., however in practice the TP cannot supply operands this rapidly. There are no bounds on the maximum duration of this interval. It is anticipated that emperical fine-turning will reduce these duration requirements by 50%.

With the exception of POLY, the AU knows how many transmissions to expect based upon a decoding of the order (IV and NT). When the correct number has been received, the AU begins execution. In the case of POLY, the TP must maintain the count of the number of coefficients sent. The last coefficient is sent to the AU with the IV field all 1's. When the last TII signal goes to 0, the TP releases the INBUS by setting PREN = 0. The Exchange Net in turn sets the TP's ENRP = 0.

The remaining signal in Table 2.3.2.1 to be discussed is IAU0. This interrupt line is used in the case in which both AU's are performing POLY orders in conjunction with two TP's and a third TP requires the use of an AU for a non-POLY order. In this case, the Exchange Net will set IAU0 = 1 to interrupt AU number 0. This unit will then return a partial result to the calling TP and next accept the pending non-POLY order. The TP holding the uncompleted POLY will initiate a request to the EN for an AU to complete the work and the request will be granted when either AU is free.

2.3.3 Implementation

Figure 2.3.3.1 is a block diagram of the V-BUS Interface. The relevant detailed logic drawings are 230-01, -02, -03, -04, -05.

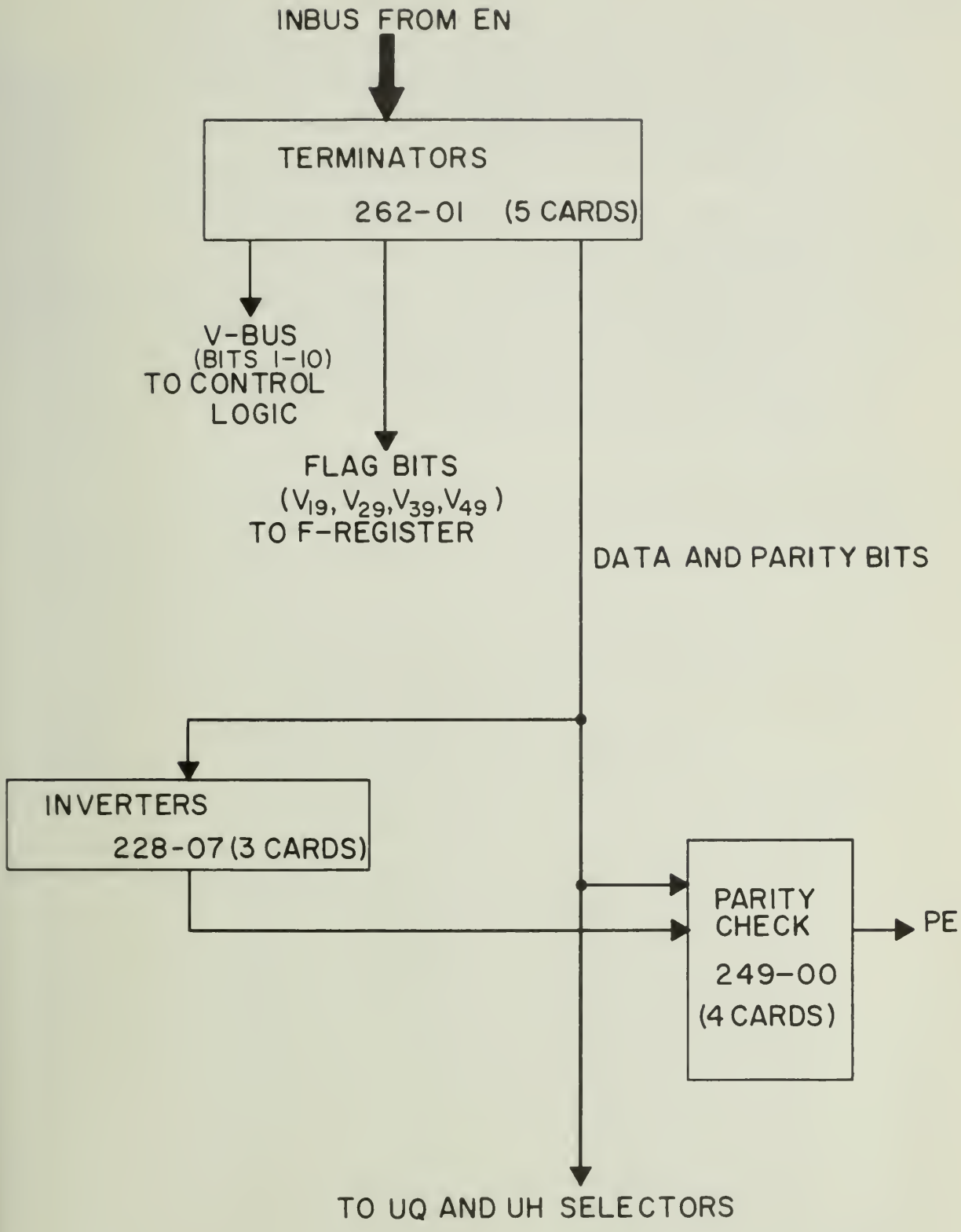


Figure 2.3.3.1 - Block Diagram of V-BUS Interface

2.3.4 PL/1 Description of Signal Names

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO V-BUS*/
/*RELEVANT DRAWING NUMBERS: 230-01,-02,-03,-04,-05*/
          DECLARE V  BIT(50);/* INPUT BUS FORM EXCHANGE */
          DECLARE ES BIT(50);/*EXCHANGE SYSTEM BUS*/
          DECLARE PE ENTRY RETURN(BIT(1))
ESDV:     /*EXCHANGE SYSTEM DIRECT TO V-BUS*/
          /*THIS SIGNAL IS ALWAYS '1'B */
PE:       PROCEDURE BIT(1);/*PARITY ERROR DETECT ACROSS BYTES
                             1-4 OF V-BUS*/
          /*RELEVANT DRAWING NUMBERS: 230-03,-04*/
          DECLARE (PE1,PE2,PE3,PE4) BIT(1),
          PE1=PE2=PE3=PE4='1'B;
          DO I= 1 TO 10; /*SUM OF 1'S IN EACH BYTE MUST BE ODD*/
          IF SUBSTR(V,10+I,1) THEN PE1= ¬PE1;
          IF SUBSTR(V,20+I,1) THEN PE2= ¬PE2;
          IF SUBSTR(V,30+I,1) THEN PE3= ¬PE3;
          IF SUBSTR(V,40,I,1) THEN PE4= ¬PE4; END;
          RETURN (PE1|PE2|PE3|PE4|);
          END;
```

2.4 XT-BUS Interface

2.4.1 General

All results and control signals from an arithmetic unit are transmitted by way of the XT-BUS (eXiT Bus). The XT-BUS interface consists of selectors, parity generation logic and line drivers.

The output of the V-BUS interface drives the OUTBUS of local exchange number 5 of the Exchange Net. The OUTBUS consists of five, 10-bit bytes. The bytes are designated 0 through 4. The bits are designated 0 through 49. The bits of the XT-BUS (XT_i) are designated 1 through 50; XT_1 corresponds to OUTBUS bit 0, etc. The byte numbering is identical for both the OUTBUS and the XT-BUS. Byte 0 is the control byte. The significance of each bit of this control byte is defined in the next section.

For transmission from an AU to a TP, the parity of each byte is generated in the AU immediately prior to being placed on the OUTBUS. The parity convention is such that the sum of 1's across all 10 bits is an odd number.

Table 2.4.1.1 summarizes the format of the XT-BUS.

OUTBUS BIT NO.	XT-BUS BIT NO.	DEFINITION
0-9	1-10	Control byte (see Tables 2.4.2.1 and 2.4.2.2).
10-17	11-18	Data bits of byte 1
18	19	Flag bit of byte 1
19	20	Parity bit of byte 1
20-27	21-28	Data bits of byte 2
28	29	Flag bit of byte 2
29	30	Parity bit of byte 2
30-37	31-38	Data bits of byte 3
38	39	Flag bit of byte 3
39	40	Parity bit of byte 3
40-47	41-48	Data bits of byte 4
48	49	Flag bit of byte 4
49	50	Parity bit of byte 4

Table 2.4.1.1 - Format of OUTBUS/XT-BUS

The XT-BUS Interface includes selectors which select the data to be placed on the OUTBUS. The selectors signals are described verbally below and defined in detail in Section 2.4.4.

Signal Name	Description
EULDXT	Select the sign of the result (SR), the exponent of the result (contents of EUL), and the first flag bit (FA_1) into XT, byte 1
UQ0DXT	Select byte 0 of the UQ-Register and the first flag bit (FA_1) into XT, byte 1.
UQ13DXT	Select bytes 1 through 3 of the UQ-Register and flag bits 2 through 4 (FA_2-FA_4) into XT, bytes 2 through 4.
UQ47DXT	Select bytes 4 through 7 of the UQ-Register and flag bits 5 through 8 (FA_5-FA_8) into XT, bytes 1 through 4.

The output of the XT Selector may be thought of as four, nine bit bytes. Parity bits are generated for each byte and become XT_{20}, XT_{30}, XT_{40} and XT_{50}. The entire 50 bits of the XT-BUS (including the 10 control bits) are gated to the Exchange System (onto the OUTBUS) under control of the signal XTDES.

2.4.2 Output from the Arithmetic Units via the XT-BUS

The structure of the control byte which is returned to a TP from an AU is shown in Table 2.4.2.1. A brief description of each signal name is given in Table 2.4.2.2. The nomenclature is consistent with that described in Department of Computer Science File No. 790.

When the AU has results to return to the calling TP, the signal UREN is set to 1 and the EN makes a path to the TP specified into the AU Processor Identification Register. This register contains the identification number of the TP that last accessed the AU. When the return path is made, ENRU (in the INBUS, Table 2.3.2.1) is set to 1. The AU may then transfer information by placing valid data on the ØUTBUS and generating the appropriate sequence of TIØ signals. An AU returns either 1 or 2 words. When the last TIØ signal goes to 0, the AU may release the ØUTBUS by setting UREN = 0. The Exchange Net will then set the AU's ENRU = 0.

When AU #0 is interrupted during a POLY order, the partial result will be returned to the TP as described above except that MCI = 1, indicating that the order has not been completed. The fact that AU#0 was executing a POLY order is transmitted to the Exchange Net by UMC = 1.

If a Unit Malfunction such as low-voltage is detected in the course of executing an AU order, UM will be set to 1. An error in the result caused by improper format of operands or results out of bounds will set the BR (Bogus Result) bit. A more specific indication of the nature the error condition is given by the flags of result.

The signal UB is set to 1 by the AU as soon as the pending AU order is decoded. Being a 1, it prevents the Exchange Net from assigning it to another AU prior to completion of the present order.

TP - EN Interface	Exchange Net ØUTBUS Bit No.	XT-BUS BIT NO.	AU - EN Interface
$\overline{\text{UREN}}$	0	1	$\overline{\text{UREN}}$
$\overline{\text{ENRP}}$	* 1 *	2	(none)
$\overline{\text{TIØ}}$	2	3	$\overline{\text{TIØ}}$
$\overline{\text{UB}} = \overline{\text{US0}}$	3	4	$\overline{\text{UB}}$
$\overline{\text{UM}} = \overline{\text{US1}}$	4	5	$\overline{\text{UM}}$
$\overline{\text{UPE}} = \overline{\text{US2}}$	5	6	$\overline{\text{UPE}}$
$\overline{\text{UMC}} = \overline{\text{US3}}$	6	7	$\overline{\text{US3}} = \overline{\text{UMC}}$
$\overline{\text{MCI}} = \overline{\text{US4}}$	7	8	$\overline{\text{US4}} = \overline{\text{MCI}}$
$\overline{\text{BR}} = \overline{\text{US5}}$	8	9	$\overline{\text{US5}} = \overline{\text{BR}}$
(none)	* 9 *	10	(none)

*Indicates that this line does not go through the Exchange Net.

Table 2.4.2.1 - ØUTBUS Control Bit Assignment

Name	Description	Equivalent Name in AU Documentation if Different
UREN*	Unit Requests Exchange Net	Exchange Request
ENRP*	Exchange Net Reply to Processor	----
TIØ	Transfer Information Out	Out Info Ready
UB	Unit Busy	----
UM	Unit Malfunction	
UPE	Unit Parity Error	Parity Error
UMC	Unit Multi-Cycle	Multi-Cycle in Progress
MCI	Multi-Cycle Interrupt (to notify TP that the AU it has been using has been interrupted).	----
BR	Bogus Result (Indicates to the TP that the result is incorrect. The TP determines the nature of the error from the flags of incorrect result).	----
US_n*	Unit Status, Bit n (These standard signals are assigned to specific status signals as shown in Table 2.4.2.1	----

*Standard signals used in the control bytes of all Processors and Units. See DCS File No. 790.

Table 2.4.2.2

Description of OUTBUS Control Signals

When improper parity occurs in any words of the operands, a flip-flop is set but loading of the operands continues. On completion of the loading, the AU does not, however, begin executing the order. It rather jumps to the ØUTBUS sequence and returns a one word pseudo result (all 0's) to the calling TP with the UPE bit set to 1. The AU will then reset as if the order had been successfully completed. The TP will initiate appropriate recovery action which may include reattempting execution of the order.

2.4.3 Implementation

Figure 2.4.3.1 is a block diagram of the XT-BUS Interface. The relevant detailed logic drawings are 240-01, -02, -03, -04. The selector gates are implemented with two boards of 235-00 (6 gates per board) and three boards of 294-00 (8 gates per board).

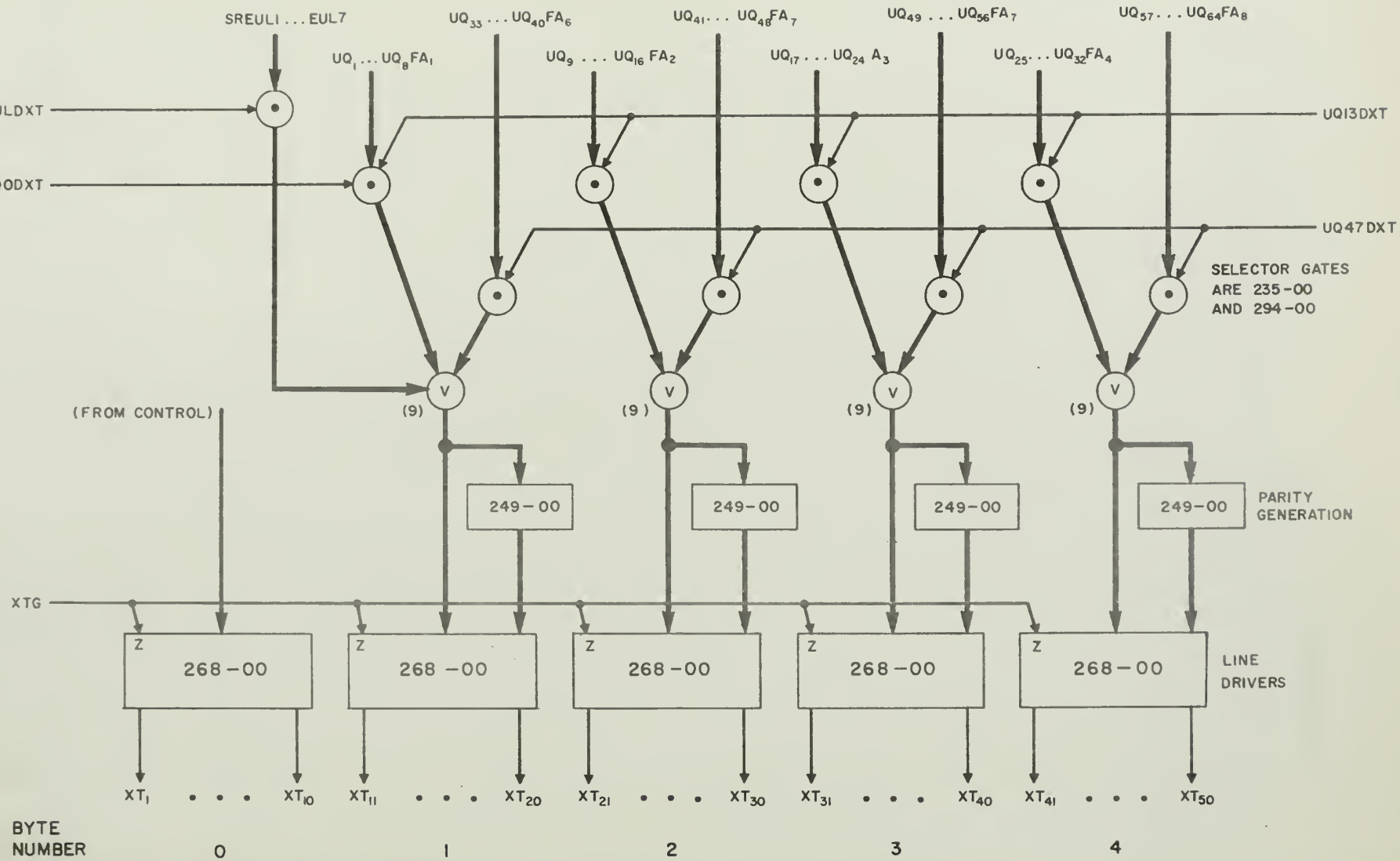


FIGURE 2.4.3.1 BLOCK DIAGRAM OF XT-BUS INTERFACE

2.4.4 PL/1 Description of Signal Names

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO XT-BUS*/
/*RELEVANT DRAWING NUMBERS: 240-01,-02,-03,-04*/
         DECLARE XT BIT(50);/*OUTPUT OF XT (EXIT) BUS SELECTOR*/
         DECLARE (PARBIT1,PARBIT2,PARBIT3,PARBIT4) ENTRY
                 RETURN(BIT(1)),TEMP BIT(1);
EULDXT:  /* EUL REGISTER TO XT-BUS BYTE 1*/
         PROCEDURE;
         SUBSTR(XT,11,9)=SR||EUL;
         SUBSTR(XT,20,1)=PARBIT1;
         END;
PARBIT1: PROCEDURE BIT(1); K=10; GO TO START;
PARBIT2: PROCEDURE BIT(1); K=20; GO TO START;
PARBIT3: PROCEDURE BIT(1); K=30; GO TO START;
PARBIT4: PROCEDURE BIT(1); K=40;
START:   TEMP='1'B;
         DO I=1 TO 9; /* SUM OF 1'S ACROSS 10 BITS IS ODD*/
         IF SUBSTR(XT,K+I,1) THEN TEMP=¬TEMP; END;
         RETURN (TEMP);
         END;
UQ0DXT:  /* UQ-REGISTER BYTE 0 TO XT BUS BYTE 1 */
         PROCEDURE;
         SUBSTR(XT,11,9) =SUBSTR(UQ,1,8)||SUBSTR(FA,1,1);
         SUBSTR(XT,20,1) =PARBIT1;
         END;
UQ13DXT: /* UQ-REGISTER BYTES 1-3 TO XT BUS BYTES 2-4*/
         PROCEDURE;
         SUBSTR(XT,21,9)=SUBSTR(UQ,9,8)||SUBSTR(FA,2,1);
         SUBSTR(XT,30,1)=PARBIT2;
         SUBSTR(XT,31,9)=SUBSTR(UQ,17,8)||SUBSTR(FA,3,1);
         SUBSTR(XT,40,1)=PARBIT3;
         SUBSTR(XT,41,9)=SUBSTR(UQ,25,8)||SUBSTR(FA,4,1);
         SUBSTR(XT,50,1)=PARBIT4;
         END;
UQ47DXT: /* UQ-REGISTER BYTES 4-7 TO XT-BUS BYTES 1-4*/
         PROCEDURE;
         SUBSTR(XT,11,9) =SUBSTR(UQ,33,8)||SUBSTR(FA,5,1);
         SUBSTR(XT,20,1) =PARBIT1;
         SUBSTR(XT,21,9) =SUBSTR(UQ,41,8)||SUBSTR(FA,6,1);
         SUBSTR(XT,30,1) =PARBIT2;
         SUBSTR(XT,31,9) =SUBSTR(UQ,49,8)||SUBSTR(FA,7,1);
         SUBSTR(XT,40,1) =PARBIT3;
         SUBSTR(XT,41,9) =SUBSTR(UQ,57,8)||SUBSTR(FA,8,1);
         SUBSTR(XT,50,1) =PARBIT4;
         END;
XTDES:   /* EXIT BUS SELECTOR OUT DIRECT TO EXCHANGE SYSTEM*/
         PROCEDURE;
         ES=XT
         END;
```

2.5 Signed-Digit Subtracters

2.5.1 General

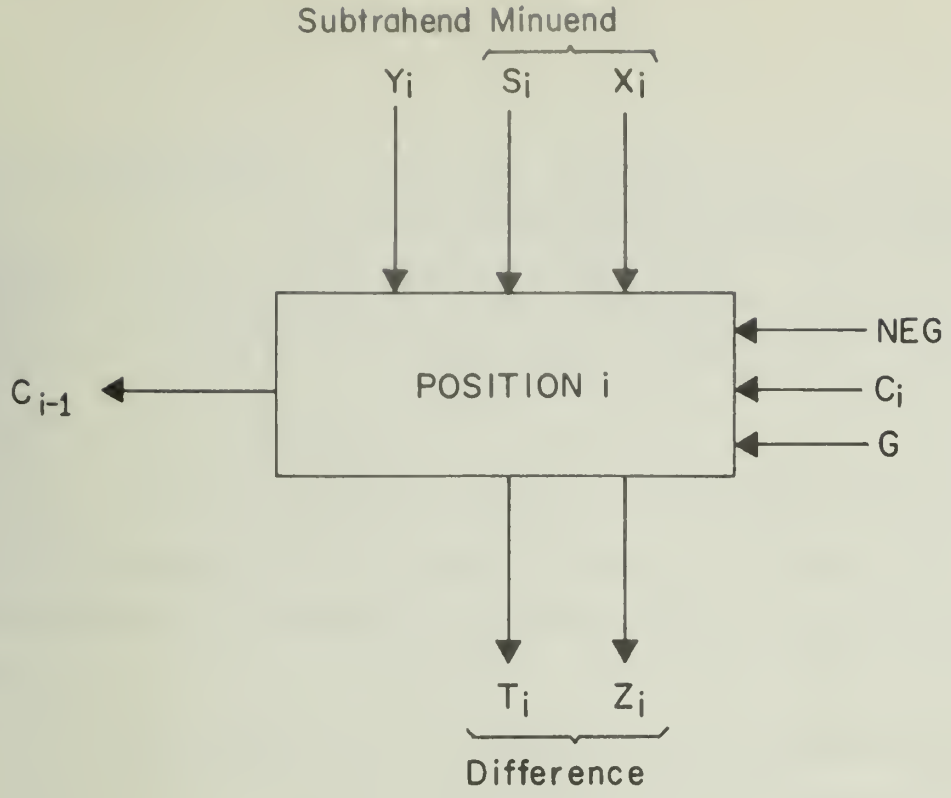
Multiplication is substantially accelerated by the use of a carry-save adder which eliminates carry or borrow propagation until a terminal step. This consideration is central to the design of the adder-subtracter of Illiac III.

The "adder" used in Illiac III is actually a signed-digit subtracter: it includes the facility for postponing borrow propagation. As will be shown, this device will perform both addition and subtraction.

Each stage of the signed-digit subtracter (SDS), as shown in Figure 2.5.1.1 is a 3-input, 2-output device together with an interstage connection, a "NEG" control line, and the G-control. Here Y_i is a bit of the subtrahend (minuend - subtrahend = difference) in conventional binary form. S_i and X_i, together comprise the minuend digit in a redundant notation which will be referred to as SD format. Each digit of the minuend is of the form S_iX_i, where X_i is interpreted as a magnitude, 1 or 0, and S_i as a sign, 0 = +, 1 = -. The SD format digits are therefore represented as follows:

S_i	X_i	Digital Value
0	0	+0
0	1	+1
1	0	-0
1	1	-1

The output of the subtracter is in this same SD format, i.e. Z_i is the magnitude of the digit, T_i is the sign.



S_i = sign of minuend digit

X_i = magnitude of minuend digit

Y_i = subtrahend in conventional binary form

T_i = sign of difference digit

Z_i = magnitude of difference digit

NEG = control to complement T_i
If NEG = 1 then T_i is complemented, else not

G = gate on interpositional connections

C_i = interpositional connection

$$T_i = C_i \oplus NEG$$

$$Z_i = C_i \oplus X_i \oplus Y_i$$

$$C_{i-1} = (S_i X_i \vee \overline{X}_i Y_i) \quad G$$

$$C_i = (S_{i+1} X_{i+1} \vee \overline{X}_{i+1} Y_{i+1}) \quad G$$

Figure 2.5.1.1 - Typical Position of a Signed-Digit Subtracter

C_i and C_{i-1} are interstage connections. As may be seen from the logical equation, these are <u>not</u> propagating borrows.

G is a gate signal which is normally "1" when the SDS is adding or subtracting. When, however, it is necessary to pass data through SDS without changing its form, G is set to "0".

SDS1, SDS2, SDS3 and SDS4 on the block diagram, Figure 2.1.1.1 are signed-digit subtracters.

NEG is a control signal which when equal "1" complements T_i, thus negating the output of the subtracter. Consider a subtracter composed of several of the SDS stages shown in Figure 2.5.1.1. Let

Y = the value of the subtrahend in conventional form.
X* = the value of the minuend in SD format.
Z* = the value of the difference in SD format.

With NEG = "0", the device is truly a subtracter and Z* = X* - Y. With NEG = "1", the output is negated, thus Z* = -(X* - Y). Note now that if complementing circuits are added to the S_i inputs so that <u>both</u> X* and Z* may be negated, Z* = -(-X* - Y) = X* + Y, and the device is adding. Actually the negating circuits for X are not included in the subtracter, <u>per se</u>. The same result is achieved by gating the complement of S from the S-input register, or, when the subtracters are cascaded, by negating the previous stage output.

Before the results formed by an SDS can be returned to a TP they must be assimilated into conventional form. This assimilation actually requires a borrow propagation and is described in Section 2.7, Propagation Logic.

2.5.2 Implementation

Figure 2.5.2.1 is a logic drawing of one half of the 1018-275-00 PC card with which the SDS is actually implemented. The other half is identical. The signal names are noted and are defined in the same way as those of Figure 2.5.1.1 except that NEG is replaced by N; C_i is replaced by CIN, and C_{i-1} is replaced by COUT.

Additional hardware is appended to position 1 (the high-order position) of each subtracter. This hardware is necessary to correct a bogus overflow condition which arises from the redundant nature of the signed-digits. The problem arises when the input to position 1 is $\bar{1}$, i.e. when $S_1 = 1$, $X_1 = 1$. In this case, without the correction the output of position 1 and position 0 will be $Z_0^* = 1$, $Z_1^* = \bar{1}$ or $Z_0^* = \bar{1}$, $Z_1^* = 1$. But since the 0th position is not implemented a high power of 2 would be lost. The result would be algebraic equivalent, however, if the Z_0^* were lost and the sign of Z_1^* complement. This is what is done. If $S_1X_1 = 1$, then T_1 is complemented. The logic equation for the sign bit output for position 1 is therefore

$$T_1 = (N \oplus CIN_1) \oplus S_1X_1$$

It is important to implement this logic without introducing additional propagation delay in the first position. We now denote the T_1 output of the subtracter card (1018-275-00) as TI_1 (T Intermediate). As TI_1 is formed in the 275 logic, $\overline{TI_1}$, S_1X_1 and $\overline{S_1X_1}$ are formed in the special logic:

$$T_1 = TI_1 \ \overline{S_1X_1} \ v \ \overline{TI_1} \ S_1X_1 \qquad S_1X_1 = TI_1 \oplus S_1X_1$$

$$\bar{T}_1 = TI_1 \ S_1X_1 \ v \ \overline{TI_1} \ \overline{S_1X_1}$$

The bogus overflow correction logic is shown in Figure 2.5.2.2.

Detailed logic for the signed-digit subtracters is shown in Drawing Nos. 250-01, -02, -03, -04, -05, -06, -07, -08, -09, -10, -11.

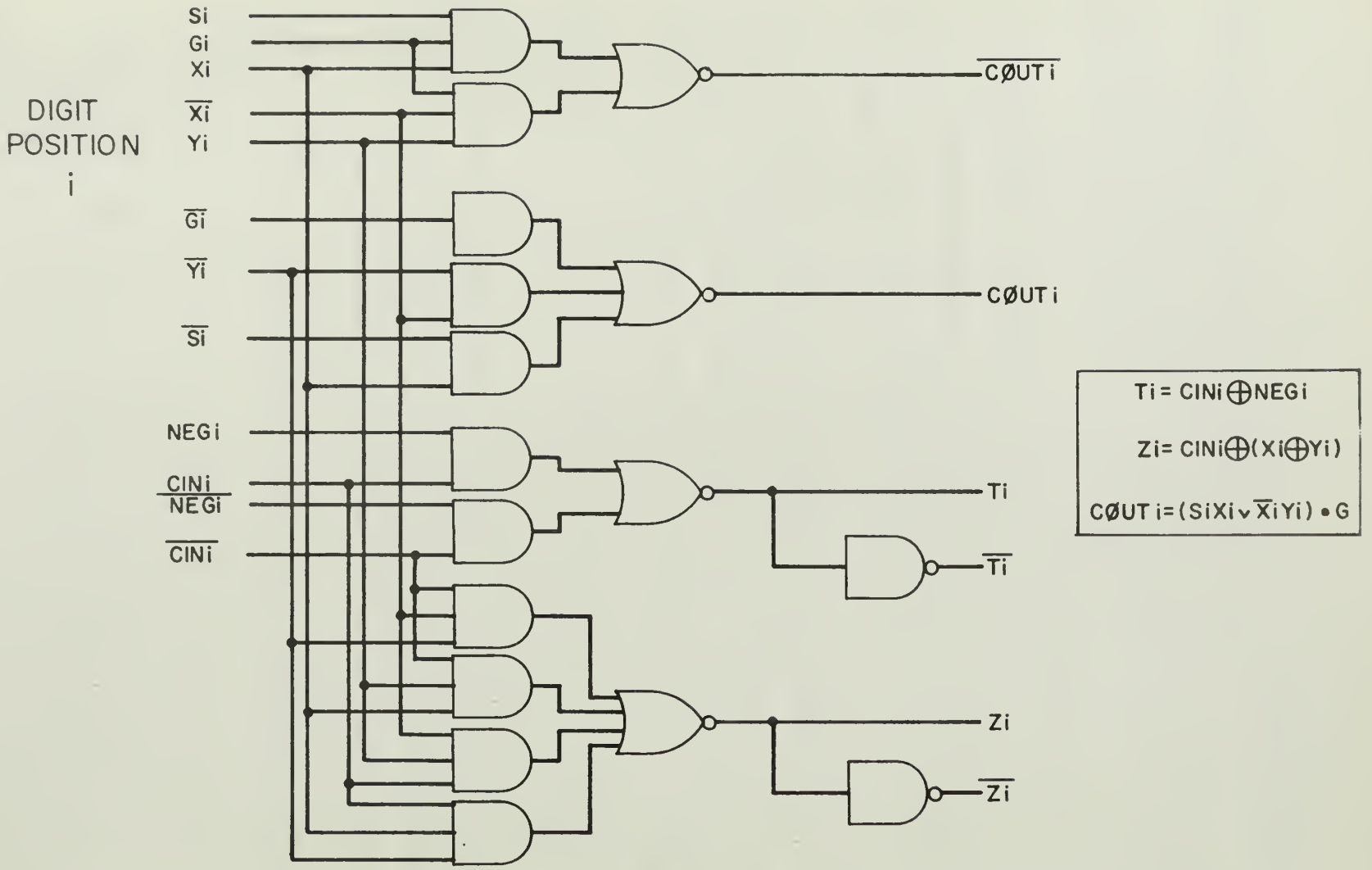
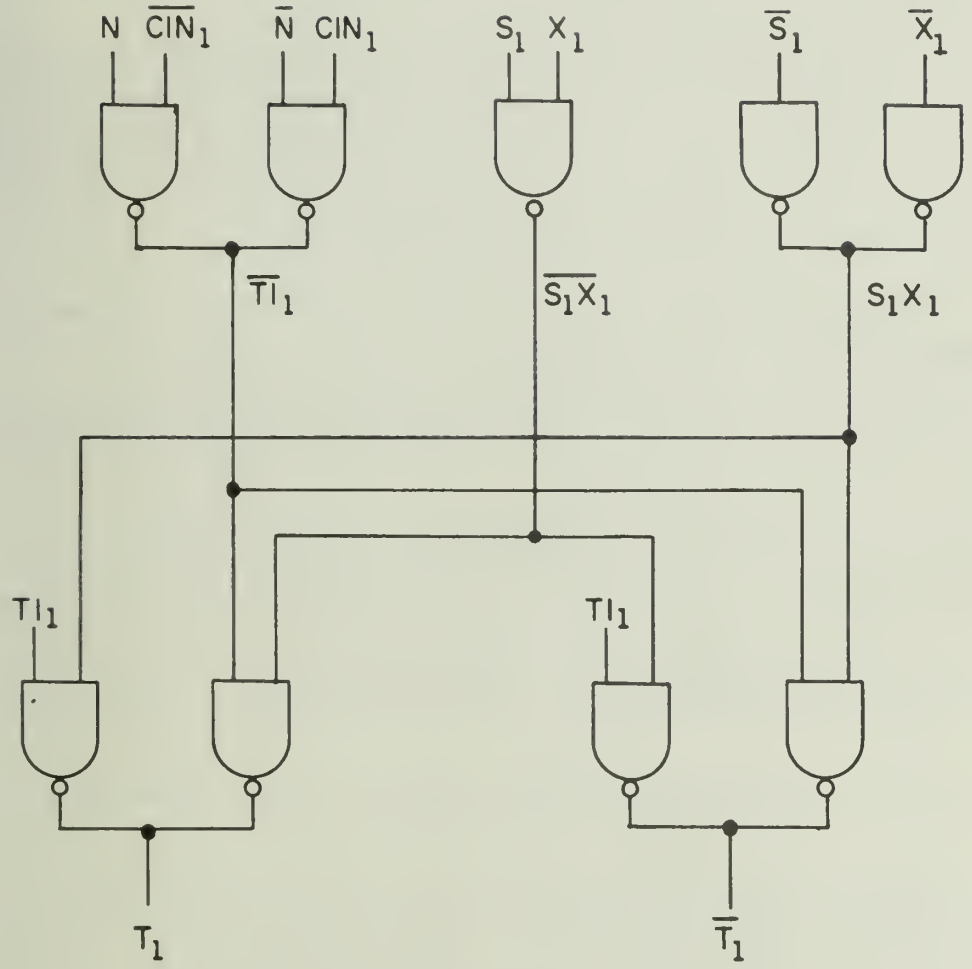


Figure 2.5.2.1 - Logic for One Position of Signed-Digit Subtracter



NOTES: 1. ALL NANDS ARE 241-00

2. TI_1 IS THE T_1 OUTPUT FROM 275-00 FOR 1st POSITION.

3. DETAILED LOGIC ON DRAWING NO. 250-11

Figure 2.5.2.2 - SDS Bogus Overflow Correction Logic

2.7 Propagation Logic

2.7.1 General

A result produced by the signed-digit subtracters is in a redundant form which must be assimilated into a conventional form before being transferred to the UQ-Register and returned to the calling TP.

This assimilation is accomplished via the propagation logic and subtracter 4. The inputs to the propagation logic are the outputs of subtracter 1, T1i and Z1i (i=1, 2, ..., 64). The outputs of the propagation logic are the P_i bits. These are gated into Y_4, (the Y-input to subtracter 4) by selecting the signal PDY4 of the M-Shift Array.

During assimilation control signals G2, G3, G4 of subtracters 2,3,4, respectively, are set at '0'. This setting permits the magnitude bits of the output of subtracter 1, Z1 to propagate through subtracters 2 and 3 unaltered. Subtracter 4 with G4 = 0 produces the exclusive OR between the Z (magnitude) bits and the output of the propagation logic, P bits, to produce the assimilated result at the output of subtracter 4, i.e. on Z4. The T4 (sign) bits have no significance. The Z4 output is then gated into the LM-Register.

The P bits formed by the propagation logic are defined by the following recursive relationship:

$$P_{i-1} = P_i \, \overline{Z1_i} \vee Z1_i \; T1_i$$

with $i = 64, 63, \ldots, 1$; $P_{64} = 0$

Now consider the logic equations for the signed-digit subtracter a as defined in Figure 7.3.7.4.1. For the fourth subtracter

$$Z4_i = C_i \oplus (X4_i \oplus Y4_i)$$

$$C_i = 0 \text{ with } G4 = 0$$

Thus $$Z4_i = X4_i \oplus Y4_i$$

But since PDY4 is set, $Y4_i = P_i$ and since G2 = G3 = 0 then $X4_i = Z1_i$, thus

$$Z4_i = Z1_i \oplus P_i.$$

Z4 becomes the result in conventional form.

The P_i bits are defined recursively, which implies that we are involved with a propagation, actually a borrow propagation. The most straightforward implementation would merely allow the borrows to ripple to the left. Once P_i was determined, P_{i-1} would be known in two collector delays using Illiac III DTL logic. Assuming a conservative estimate of 20ns per collector delay, the worst case, the formation of all 64 bits would require 63 x 40 = 2520 ns, or about 2.5 usec. This is an unacceptable figure.

The situation may be greatly improved by the use of lookahead. The techniques involves the expansion of the equation for P_i, for example,

$$P_{63} = P_{64}\ \overline{Z1_{64}}\ v\ Z1_{64}\ T1_{64}$$

$$P_{62} = P_{63}\ \overline{Z1_{63}}\ v\ Z1_{63}\ T1_{63}$$

Substituting the first equation into the second yields:

$$P_{62} = P_{64}\ \overline{Z1_{63}}\ \ \overline{Z1_{64}}\ v\ Z1_{64}\ T1_{64}\ \overline{Z1_{63}}\ v\ Z1_{63}\ T1_{63}$$

In this new form, assuming P_{64} is given, both P_{62} and P_{63} may be formed in parallel. This technique may be continued, however, successive logic equations become increasingly complex and shortly reach a limit set by hardware constraints.

In Illiac III full lookahead is generated across groups of 4 bits. Lookahead at a higher level is then performed across groups of these 4 bit groups. Finally lookahead at a third level is used across the second level groups. We will now elaborate on this.

2.7.2 First-Level Logic Equations

Assume that P_{n+1} is known and that we wish to determine P_n, P_{n-1}, P_{n-2} and P_{n-3} in parallel. Let T_i stand for Tl_i and Z_i stand for Zl_i, then

$$P_n = T_{n+1}\, Z_{n+1} \vee \overline{Z_{n+1}}\, P_{n+1}$$

$$P_{n-1} = T_n Z_n \vee \overline{Z_n}\, T_{n+1}\, Z_{n+1} \vee \overline{Z_n}\, \overline{Z_{n+1}}\, P_{n+1}$$

$$P_{n-2} = T_{n-1}\, Z_{n-1} \vee \overline{Z}_{n-1}\, T_n Z_n \vee \overline{Z}_{n-1}\, \overline{Z}_n\, T_{n+1}\, Z_{n+1}$$

$$\vee\ \overline{Z_{n-1}}\, \overline{Z_n}\, \overline{Z_{n+1}}\, P_{n+1}$$

$$P_{n-3} = T_{n-2}\, Z_{n-2} \vee \overline{Z}_{n-2}\, T_{n-1}\, Z_{n-1} \vee \overline{Z}_{n-2}\, \overline{Z}_{n-1}\, T_n\, Z_n$$

$$\vee\ \overline{Z}_{n-2}\, \overline{Z}_{n-1}\, \overline{Z}_n\, \overline{T}_{n+1}\, Z_{n+1} \vee \overline{Z}_{n-2}\, \overline{Z}_{n-1}\, \overline{Z}_n\, \overline{Z}_{n+1}\, P_{n+1}$$

$$P_{n-3} = \underbrace{T_{n-2}\, Z_{n-2} \vee \overline{Z}_{n-2} \vee Z_{n-2}\, T_{n-1}\, Z_{n-1} \vee \overline{Z}_{n-2}\, \overline{Z}_{n-1}\, T_n\, Z_n}$$

$$\underbrace{\vee\ \overline{Z}_{n-2}\, \overline{Z}_{n-1}\, \overline{Z}_n\, T_{n+1}\, \overline{Z}_{n+1}}$$

$$\downarrow$$

$$PG_{n-3}$$

$$\underbrace{\vee\ \overline{Z}_{n-2}\, \overline{Z}_{n-1}\, \overline{Z}_n\, \overline{Z}_{n+1}}\, P_{n+1}$$

$$\downarrow$$

$$PP_{n-3}$$

Now consider the expression for P_{n-3}. Let $P_{n-3} = PG_{n-3} \vee PP_{n-3} \; P_{n+1}$ where PG_{n-3} is the first 4 terms of P_{n-3} and PP_{n-3} is the last term less the variable P_{n+1}. PG_{n-3} will be 1 if it is <u>generated</u> by the positions P_{n-3} through Z_{n+1} and T_{n-2} through T_{n+1}. PP_{n-3} will be true if Z_{n-2} through Z_{n+1} are all zero. This condition will <u>propagate</u> P_{n+1}, i.e. if $PP_{n-3} = 1$, then $P_{n-3} = P_{n+1}$. Note that PG_{n-3} and PP_{n-3} are mutually exclusive. Furthermore neither is a function of P_{n+1}.

We now define the logic for a <u>first level lookahead group</u> to be a box with inputs and outputs as shown in Figure 2.7.2.1 and satisfying the first level logic equations derived above.

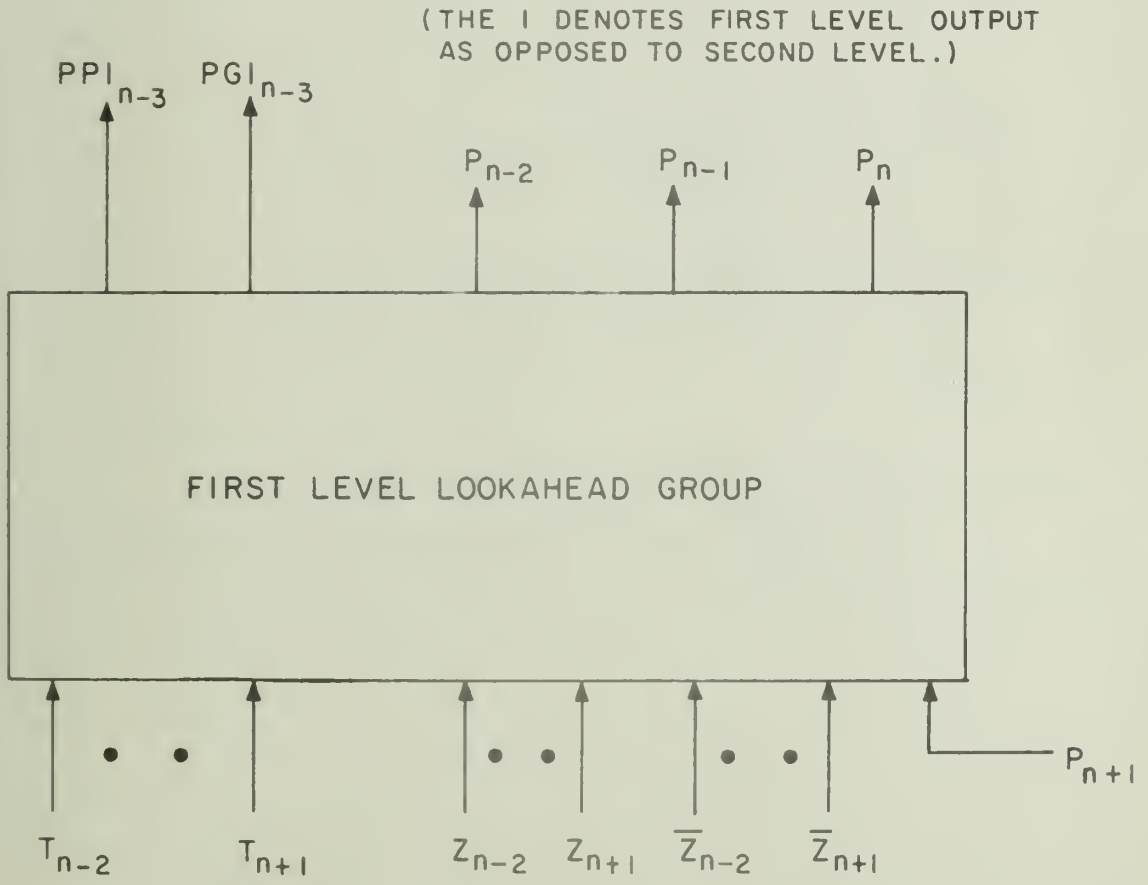


Figure 2.7.2.1 - First Level Lookahead Group

In Illiac III there is one first level group for each four bits of the 64 bit output of subtracter 1. Specific replications of the first level group are generated by letting n assume the values 3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63. $P_{64} = 0$.

2.7.3 Second Level Logic Equations

Now consider four adjacent first level groups. The PG and PP outputs will be designated PG_{n-3}, PG_{n-7}, PG_{n-11}, PG_{n-15} and PP_{n-3}, PG_{n-7}, PG_{n-11}, PG_{n-15} and PP_{n-3}, PG_{n-7}, PG_{n-11}, PG_{n-15}. As defined in the last section

$$P_{n-3} = PG_{n-3} \vee PP_{n-3}\ P_{n+1}$$

Similarly,

$$P_{n-7} = PG_{n-7} \vee PP_{n-7}\ P_{n-3} = PG_{n-7} \vee PP_{n-7}\ PG_{n-3} \vee PP_{n-7}\ PP_{n-3}\ P_{n+1}$$

$$P_{n-11} = PG_{n-11} \vee PP_{n-11}\ PG_{n-7} \vee PP_{n-11}\ PP_{n-7}\ PG_{n-3}$$

$$\vee\ PP_{n-11}\ PP_{n-7}\ PP_{n-3}\ P_{n+1}$$

$$P_{n-15} = \underbrace{PG_{n-15} \vee PP_{n-15}\ PG_{n-11} \vee PP_{n-15}\ PP_{n-11}\ PG_{n-7}}_{PG2_{n-15}}$$

$$\underbrace{\vee\ PP_{n-15}\ PP_{n-11}\ PP_{n-7}\ PG_{n-3}}_{\vee\ PP_{n-15}\ PP_{n-11}\ PP_{n-7}\ PP_{n-3}\ P_{n+1}}$$

$$PP2_{n-15}$$

As with the first level logic we now separate the propagated and generated parts of the expression for P_{n-15}. Thus,

$$P_{n-15} = PG2_{n-15} \vee PP2_{n-15}\ P_{n+1}$$

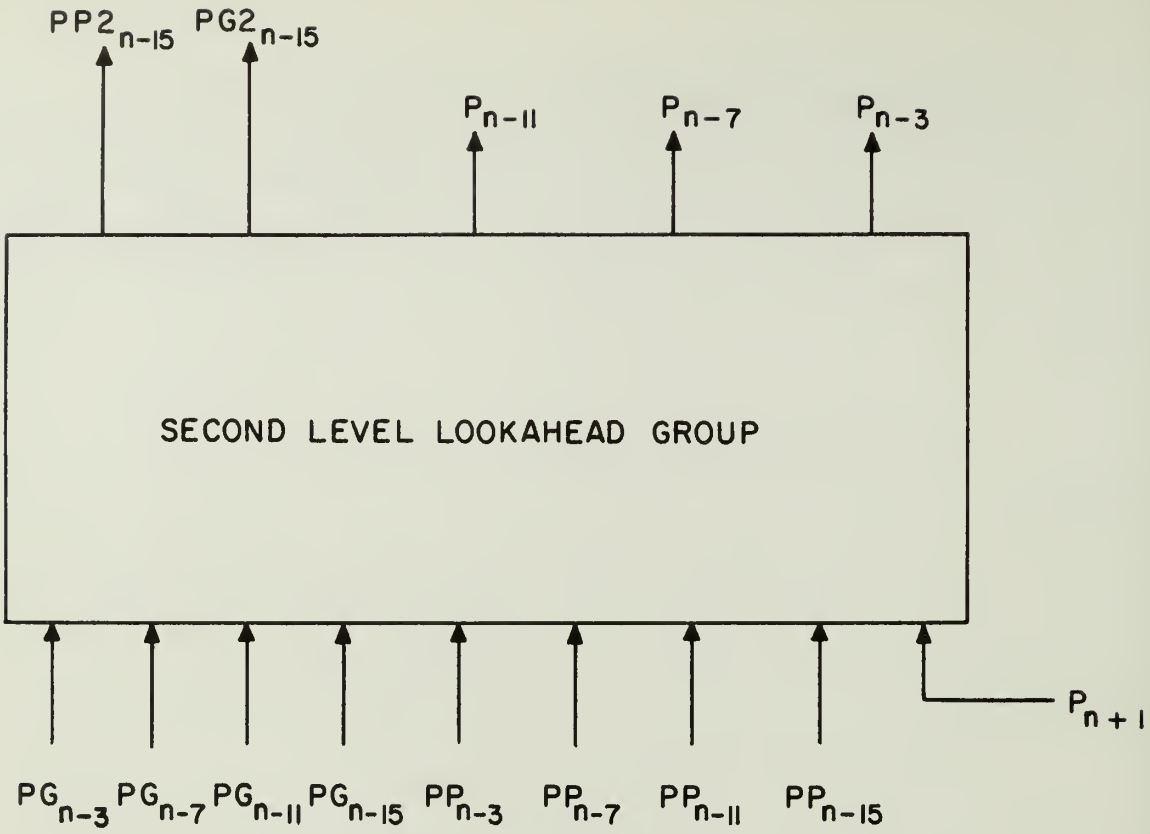


Figure 2.7.3.1 - Second Level Lookahead Group

We now define the logic for a second level lookahead group to be a box with inputs and outputs as shown in Figure 2.7.3.1 and satisfying the second level logic equations. Each second level group looks across four first level groups. Specific replications of the first level group are generated by letting n assume the values 15, 31, 47, 63.

2.7.4 Third Level Logic Equations

There are four groups at the second level. The third level looks across these to produce P_{48}, P_{32}, P_{16} and P_0. The logic equations for these are as follows:

$$P_{48} = PG2_{48} \text{ v } PP2_{48}\ P_{64}$$

$$P_{32} = PG2_{32} \text{ v } PP2_{32}\ PG2_{48} \text{ v } PP2_{32}\ PP2_{48}\ P_{64}$$

$$P_{16} = PG2_{16} \text{ v } PP2_{16}\ PG2_{32} \text{ v } PP2_{16}\ PP2_{32}\ PG2_{48} \text{ v } PP2_{16}\ PP2_{32}\ PP2_{48}\ P_{64}$$

$$P_0 = PG2_0 \text{ v } PP2_0\ PG2_{16} \text{ v } PP2_0\ PP2_{16}\ PG2_{32} \text{ v } PP2_0\ PP2_{16}\ PP2_{32}\ PG2_{48} \text{ v } PP2_0\ PP2_{48}\ PP2_{32}\ PP2_{48}\ P_{64}$$

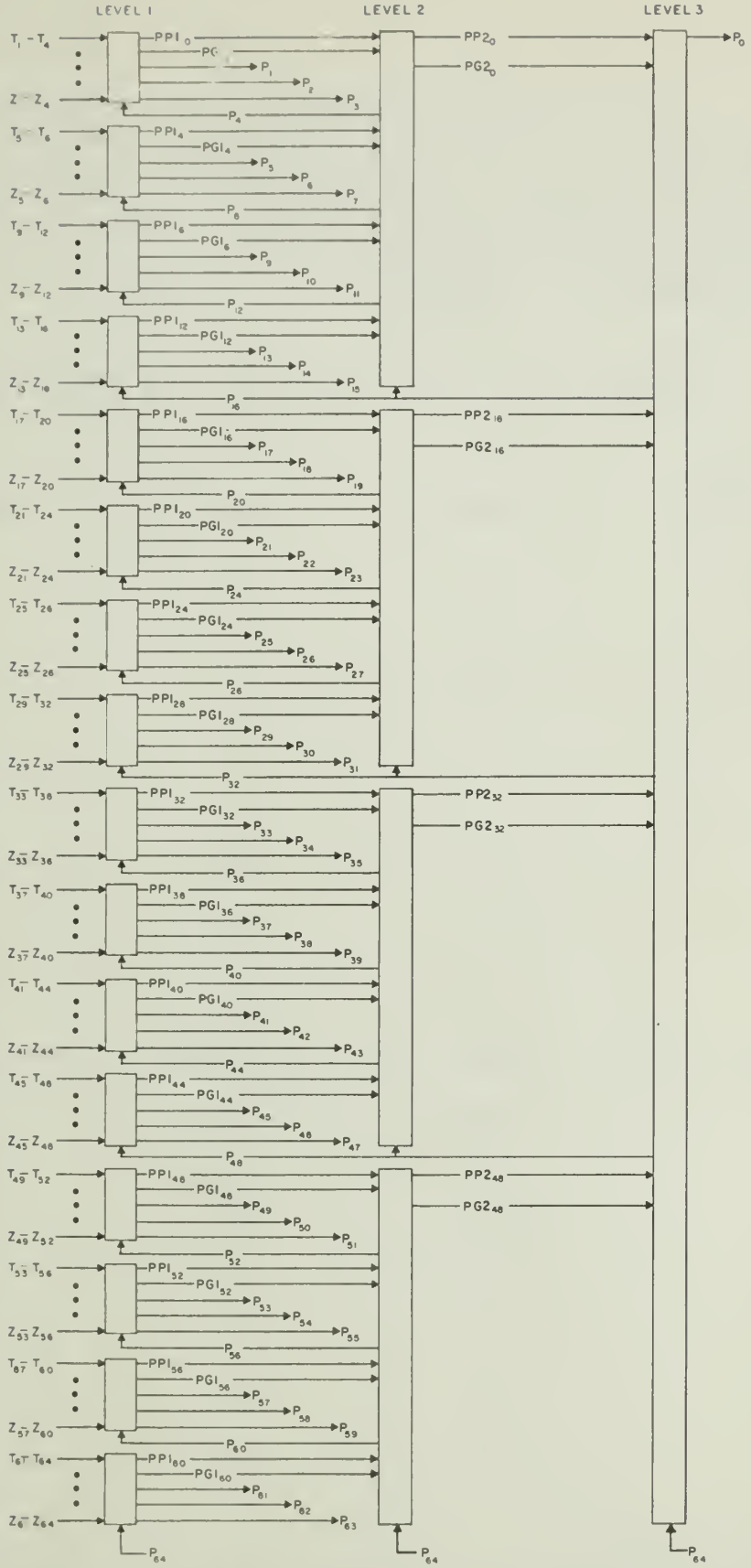
2.7.5 Implementation

Figure 2.7.5.1 is a block diagram of the entire propagation logic structure. The following table describes the operation and timing;

Step		Timing (collector dela
1	All PP1 and PPG1 signals are formed from T and Z inputs at level 1.	2
2	All PP2 and PG2 signals are formed from PP1 and PG1 inputs at level 2.	2
3	P_0, P_{16}, P_{32}, P_{64} are formed from PP2 and PG2 inputs at level 3	2
4	P_4, P_8, P_{12}, P_{16}, P_{20}, P_{24}, P_{28}, P_{32} P_{36}, P_{40}, P_{44}, P_{48}, P_{52}, P_{56} and P_{60} are formed at level 2.	2
5	The remaining P_i's are formed at level 1	2
	Total Delays	10

Note that the signals propagate from level 1 to level 3 and then back down to level 1.

The logic for all levels is implemented with the 297-03 diode-matrix board followed by the 228-05 NOR (the only board type to be driven



NOTE: P_{64} is from Multiply Extended Precision Logic (Section 2.9.3).

Figure 2.7.5.1 - Block Diagram of Propagation Logic

by diode matrix). The 228-07 is used to provide required logic inversion between levels. Figure 2.7.5.2 illustrates the 297-03. Table 2.7.5.1 gives the input pin designations for the board as used at each level; Table 2.7.5.2 gives the output pin assignment.

TABLE 2.7.5.1 - Input Assignments for 297-03 Board

INPUT PINS	LEVEL 1	LEVEL 2	LEVEL 3
2	P_{n+1}	P_{n+1}	n.c.
3	Z_{n+1}	PG_{n-3}	$PG2_{48}$
4	$\overline{Z}_{n+1}$	PP_{n-3}	$PP2_{48}$
5	T_{n+1}	n.c.	n.c.
6	Z_n	PG_{n-7}	$PG2_{32}$
8	$\overline{Z}_n$	PP_{n-7}	$PP2_{32}$
9	T_n	n.c.	n.c.
10	Z_{n-1}	PG_{n-11}	$PG2_{16}$
13	$\overline{Z}_{n-1}$	PP_{n-11}	$PP2_{16}$
14	T_{n-1}	n.c.	n.c.
15	Z_{n-2}	PG_{n-15}	$PG2_0$
17	$\overline{Z}_{n-2}$	PP_{n-15}	$PP2_0$
18	T_{n-2}	n.c.	n.c.
19	n.c.	n.c.	n.c.
20	n.c.	n.c.	n.c.
22	n.c.	n.c.	n.c.

Notes: 1. For level 1, n = 3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63.
2. P_{64} = 0 = ground.

TABLE 2.7.5.2 - Output Assignments for 297-03 Board.

OUTPUT PINS	LEVEL 1	LEVEL 2	LEVEL 3
A,B,C	$PG1_{n-3}$	$PG2_{n-15}$	P_0
E	"	"	n.c.
F	$PP1_{n-3}$	$PP2_{n-15}$	n.c.
H,J,L	P_{n-2}	P_{n-11}	P_{16}
P	"	"	n.c.
S,T	P_{n-1}	P_{n-7}	P_{48}
U	"	"	n.c.
V,X	P_n	P_{n-3}	n.c.
Y,Z	n.c.	n.c.	n.c.

See notes on Table 2.7.5.1

The numbers of the detailed logic drawings for the propagation 1 logic drawings for the propagation logic are 270-01, -02, -03, -04, -05, -06, -07.

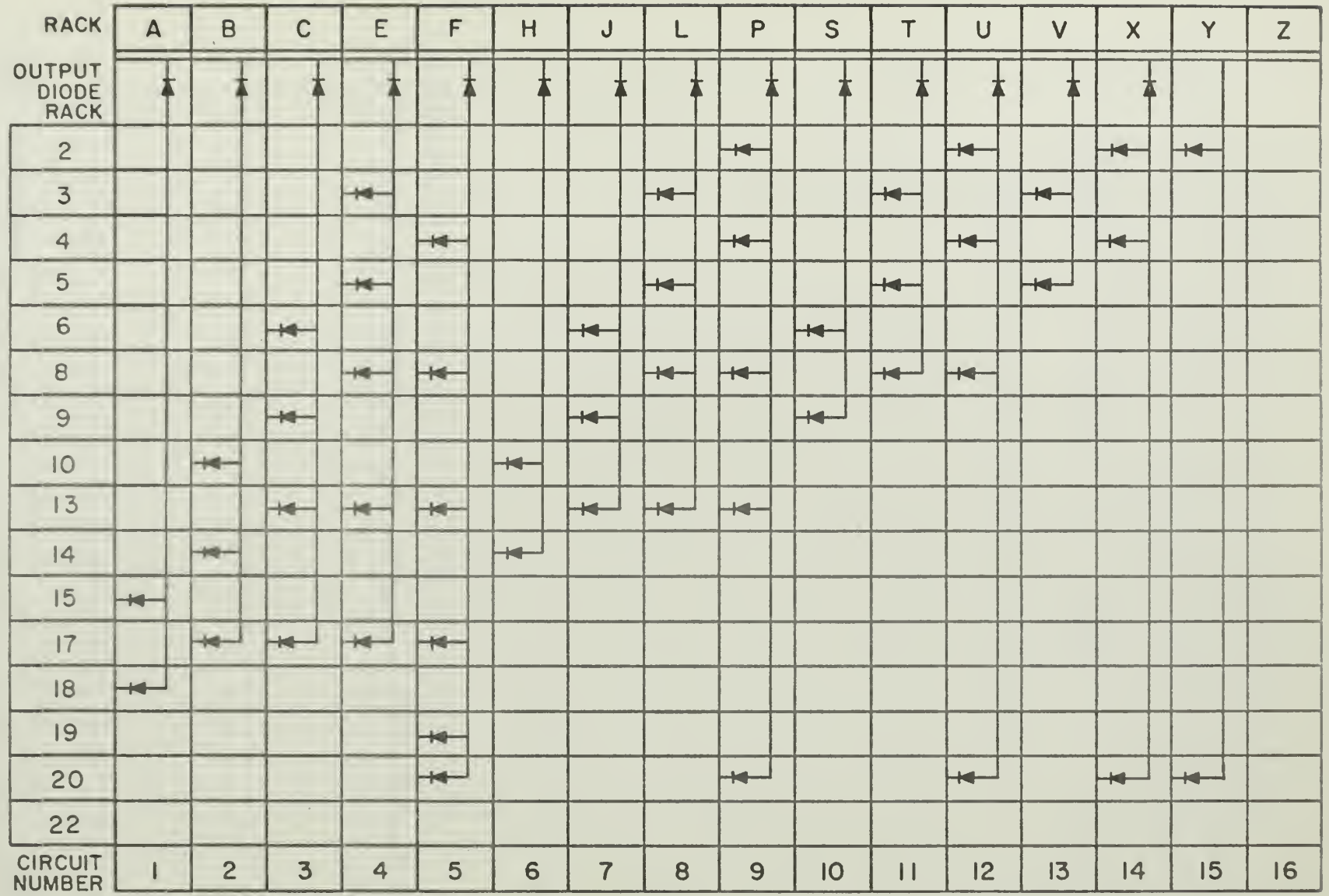


Figure 2.7.5.2 - Propagation Logic Diode Matrix Board
(297-03)

2.8 M-Shift Array

2.8.1 General

The M-Shift Array couples the output of the M-Register with the conventional ("Y") inputs to the four signed digit subtracters (see Section 2.5). The array also includes several gates which are driven by other than the M-Register, namely, PDY4, UHDY1 and the TENIn series of gates. These are defined later in this section.

The gate signals associated with the M-Shift Array and their primary use are described below:

Signal Name	Description	Primary Use
MDY1	M-Register direct (without shift) to Y input of subtracter 1.	To supply the subtrahend or addend during ADD/SUB; to supply an additional multiple of multiplicand during terminal loop of MPY.
MDY4	M-Register direct (without shift) to Y input of subtracter 4.	To supply a multiple of 1 times the contents of the M-Register during MPY/DIV.
ML1Y4	M-Register left 1 bit to Y input of subtracter 4.	To supply a multiple of 2 times the contents of the M-Register during MPY/DIV.
ML2Y3	M-Register left 2 bits to Y input of subtracter 3.	To supply a multiple of 4 times the contents of the M-Register during MPY/DIV.
ML3Y3	M-Register left 3 bits to Y input of subtracter 3	To supply a multiple of 8 times the contents of the M-Register during MPY/DIV.

Signal Name	Description	Primary Use
ML4Y2	M-Register left 4 bits to Y input of subtracter 2.	To supply a multiple of 16 times the contents of the M-Register during MPY/DIV.
ML5Y2	M-Register left 5 bits to Y input of subtracter 2.	To supply a multiple of 32 times the contents of the M-Register during MPY/DIV.
ML6Y1	M-Register left 6 bits to Y input of subtracter 1.	To supply a multiple of 64 times the contents of the M-Register during MPY/DIV.
ML7Y1	M-Register left 7 bits to Y input of subtracter 1.	To supply a multiple of 128 times the contents of the M-Register during MPY/DIV.
PDY4	The output of the Propagation Logic direct (without shift) to the Y input of subtracter 4.	To supply the P-bits to subtracter 4 where they are combined in an exclusive -OR with the magnitude bits (Z-bits) to produce an assimilated result.
UHDY1	UH-Register direct to the Y input of subtracter 1.	To supply coefficients to the partial result during POLY.

The algorithms for conversion from binary to decimal is based upon successively taking the remainder of an integer division by ten. The signal names of the form TEN_Y_ supply this constant divisor, ten, appropriately shifted, to the subtracters. The high-order 12 bits supplied to the Y inputs for each "TEN" signal is shown below. All positions to the right of position 12 are zero.

Signal Name	Bit Pattern
TENDY4	Y4:000000001010
TENL1Y4	Y4:000000010100
TENL2Y3	Y3:000000101000
TENL3Y3	Y3:000001010000
TENL4Y2	Y2:000010100000
TENL5Y2	Y2:000101000000
TENL6Y1	Y1:001010000000
TENL7Y1	Y1:010100000000

2.8.2 Implementation

Most of the M-Shift Array is implemented with the 294-00 selector board. This board provides a three way selection. The Y1 portion of the shift array requires a 4 way selection: ML7Y1, ML6Y1, MDY1, or UHDY1. A typical position of this selector is shown in Figure 2.8.2.1. The Y2 and Y3 subsections of the shift array each require only a 2 way selection: ML5Y2, ML4Y2 and ML3Y3, ML2Y3. For these, one AND of each position of the 294-00 is unused. The Y4 subsection makes use of all three inputs: ML1Y4, MDY4, PDY4.

The M-Shift Array is operated primarily by the multiplier recoder (during MPY) or the model division (during DIV). Figure 2.8.2.2 illustrates the correspondence between the signals multiply and divide signals such as MY128X, DV128X and the M-Shift Array control signals such as ML7Y103 and ML7Y147. Note that ML7Y103 and ML7Y147 taken together constitute ML7Y1. The drivers shown in this figure implement equations of the form ML7Y1 = MY128X v DV128X. The abbreviations MY and DV denote multiply and divide, respectively. The "128X" denotes 128 times (X) the contents of the M register. A shift of 7 places to the left corresponds to multiplication by 128, i.e. by 2^7.

The relevant detailed logic drawings are 280-01, -02, -03, -04, -05, -06, -07, -08, -09, -10, -11. The logic for TENL1Y4 and TENDY4 is shown on Drawing Number 250-11. The drivers for the M-Shift Array are shown on Drawing Numbers 250-09, -10 in conjunction with drivers for the control of the subtracter cascade. On logic drawings in the 280- series both signals will be found denoted M_i or M_i^*. The signals (the true output of the ith position of the M-Register) are logically equivalent. M_i is taken from the true side of the 260-00 flip-flop of the M-Register. M_i^* is taken from the output of inverters which are driven by the complement side of the M-Register. This scheme was adopted merely to provide additional drive from the M-Register.

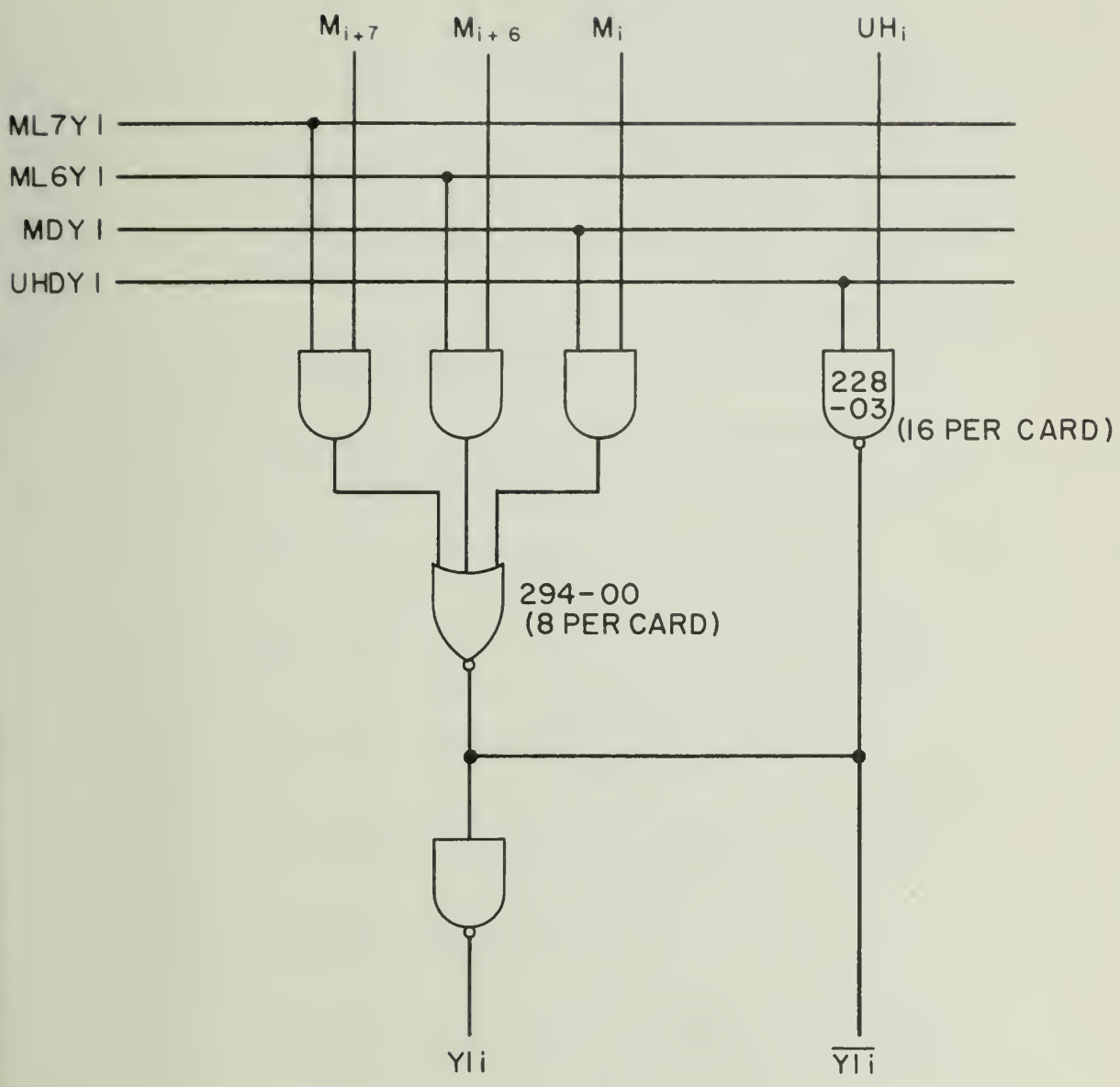
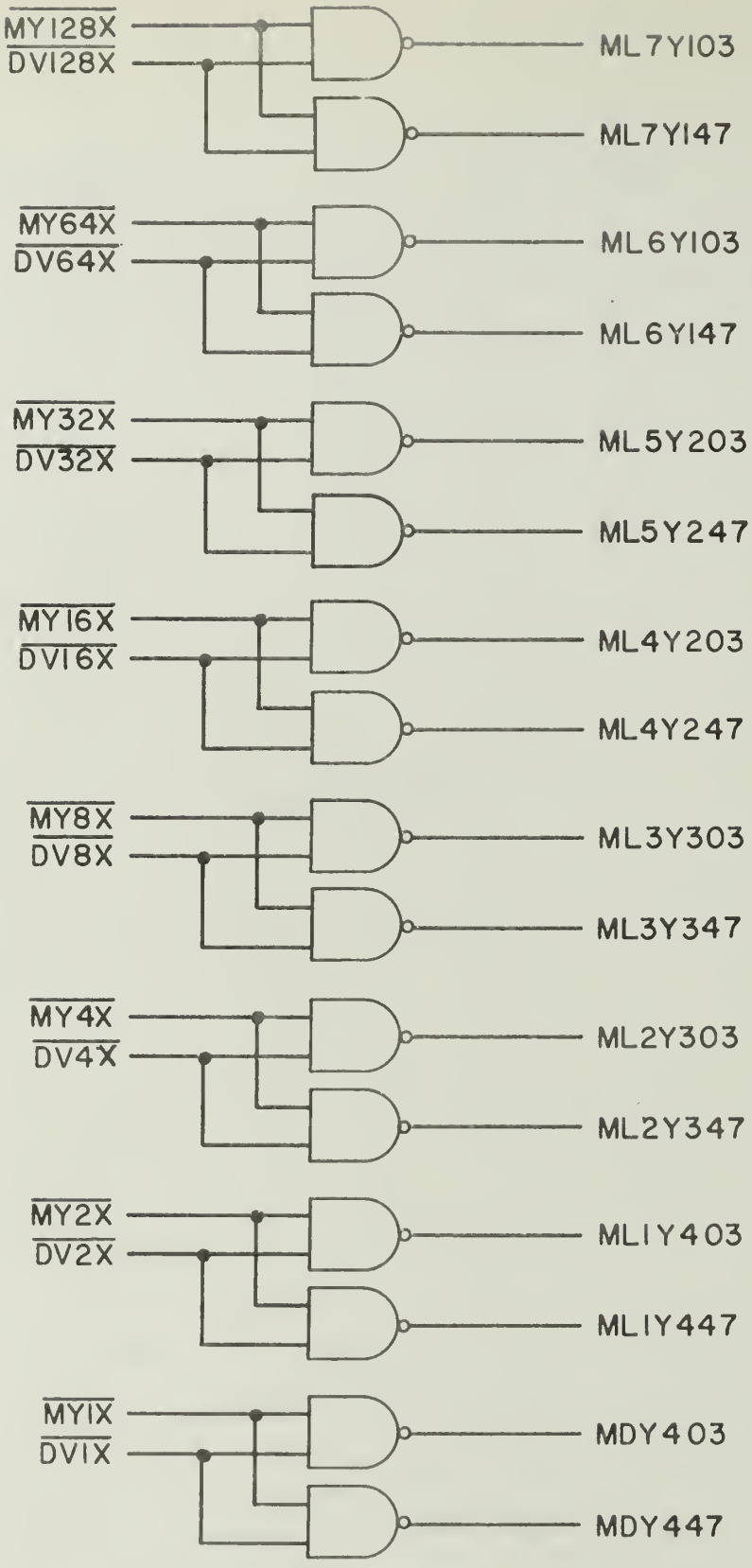


Figure 2.8.2.1 - Typical Position of Yl Subsection of the M-Shift Array



NOTE: ALL GATES ARE 214-00

Figure 2.8.2.2 - Drivers for the M-Shift Array

2.8.3 PL/1 Description of Signal Names

```
*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO M-SHIFT ARRAY */
*RELEVANT DRAWING NUMBERS: 280-01,-02,-03,-04,-05,-06,-07,-08,
                           -09,-10,-11 AND 250-09,-10,-11*/
        DECLARE M BIT(64); /*TRUE OUTPUT OF M-REGISTER. POSITIONS
                             1-8 NOT IMPLEMENTED.*/
        DECLARE (Y1,Y2,Y3,Y4) BIT(64); /*TRUE OUTPUTS OF M-SHIFT
                                         ARRAY. Y1 THRU Y4 DRIVE
                                         SUBTRACTER 1 THRU 4,
                                         RESPECTIVELY. */
        DECLARE MINJECTONE BIT(1); /* A CONTROL SIGNAL: IF THIS
                SIGNAL IS '1'B THEN 1'S ARE INJECTED IN THE HIGH
                ORDER POSITIONS OF Y1-Y4  WHICH ARE NOT DRIVEN
                BY BITS FROM THE M-REGISTER.*/
        DECLARE T BIT(1): /* A TEMPORARY STORAGE FOR A BIT - NO
                             ACTUAL HARDWARE EQUIVALENT */
        DECLARE P BIT(64):/*OUTPUT OF PROPAGATION LOGIC*/
Y103:   /*SELECT M DIRECT (NO SHIFT) TO Y1, BYTES 0-3 */
        PROCEDURE;
        SUBSTR(Y1,1,32)='00000000'B||SUBSTR(M,9,24);
        END;
Y147:   /*SELECT M DIRECT (NO SHIFT) TO Y1, BYTES 4-7 */
        PROCEDURE;
        SUBSTR(Y1,33,32)=SUBSTR(M,33,32);
        END;
Y403:   /*SELECT M DIRECT (WITHOUT SHIFT) TO Y4, BYTES 0-3*/
        PROCEDURE;
        T=MINJECTONE
        SUBSTR(Y4,1,32)=T||T||T||T||T||T||T||T||SUBSTR(M,9,24);
        END;
447:    /*SELECT M DIRECT (WITHOUT SHIFT) TO Y4, BYTES 0-3*/
        PROCEDURE;
        SUBSTR(Y4,33,32)=SUBSTR(M,33,32);
        END;
Y403:   /*SELECT M LEFT 1 BIT TO Y4, BYTES 0-3 */
        PROCEDURE;
        T=MINJECTONE
        SUBSTR(Y4,1,32)=T||T||T||T||T||T||T||T||SUBSTR(M,9,25);
        END;
Y447:   /*SELECT M LEFT 1 BIT TO Y4, BYTES 4-7 */
        PROCEDURE;
        SUBSTR(Y4,33,32)=SUBSTR(M,33,31)||'0'B;
        END;
Y303:   /*SELECT M LEFT 2 BITS TO Y3, BYTES 0-3 */
        PROCEDURE;
        T=MINJECTONE;
        SUBSTR(Y3,1,32)=T||T||T||T||T||T||T||T||SUBSTR(M,9,26);
        END;
347:    /*SELECT M LEFT 2 BITS TO Y3, BYTES 4-7 */
        PROCEDURE;
        SUBSTR(Y3,33,32)=SUBSTR(M,35,30)||'00'B;
        END;
303:    /*SELECT M LEFT 3 BITS TO Y3, BYTES 0-3 */
        PROCEDURE;
        T=MINJECTONE;
        SUBSTR(Y3,1,32)=T||T||T||T||T||T||T||T||SUBSTR(M,9,27);
        END;
347:    /*SELECT M LEFT 3 BITS TO Y3, BYTES 4-7*/
```

```
            PROCEDURE;
            SUBSTR(Y3,33,32)=SUBSTR(M,36,29)||'000'B
            END;
ML4Y203:    /*SELECT M LEFT 4 BITS TO Y2, BYTES 0-3 */
            PROCEDURE;
            T=MINJECTONE
            SUBSTR(Y2,1,32)=T||T||T||T||SUBSTR(M,9,28);
            END;
ML4Y247:    /*SELECT M LEFT 4 BITS TO Y2, BYTES 4-7*/
            PROCEDURE;
            SUBSTR(Y2,33,32)=SUBSTR(M,37,28)||'0000'B;
            END;
ML5Y203:    /*SELECT M LEFT 5 BITS TO Y2, BYTES 0-3 */
            PROCEDURE;
            T=MINJECTONE;
            SUBSTR(Y2,1,32)=T||T||T||SUBSTR(M,9,29);
            END;
ML5Y247:    /*SELECT M LEFT 5 BITS TO Y2, BYTES 4-7*/
            PROCEDURE;
            SUBSTR(Y2,33,32)=SUBSTR(M,38,27)||'00000'B;
            END;
ML6Y103:    /*SELECT M LEFT 6 BITS TO Y1, BYTES 0-3 */
            PROCEDURE ;
            T=MINJECTONE
            SUBSTR(Y1,1,32)=T||T||SUBSTR(M,9,30);
            END;
ML6Y147:    /*SELECT M LEFT 6 BITS TO Y1, BYTES 4-7 */
            PROCEDURE;
            SUBSTR(Y1,33,32)=SUBSTR(M,39,26)||'000000'B;
            END;
ML7Y103:    /*SELECT M LEFT 7 BITS TO YL, BYTES 0-3 */
            PROCEDURE;
            T=MINJECTONE;
            SUBSTR(Y1,1,32)=T||SUBSTR(M,9,31);
            END;
ML7Y147:    /*SELECT M LEFT 7 BITS TO Y1, BYTES 4-7 */
            PROCEDURE;
            SUBSTR(Y1,33,32)=SUBSTR(M,40,25)||'0000000'B;
            END;
PDY403:     /*SELECT OUTPUT OF PROPAGATION LOGIC DIRECT (WITHOUT
              SHIFT) TO Y4, BYTES 0-3*/
            PROCEDURE;
            SUBSTR(Y4,1,32)=SUBSTR(P,1,32);
            END;
PDY447:     /*SELECT OUTPUT OF PROPAGATION LOGIC DIRECT (WITHOUT
              SHIFT) TO Y4, BYTES 4-7*/
            PROCEDURE;
            SUBSTR(Y4,33,32)=SUBSTR(P,33,32);
            END;
TENDY4:   /*SELECT THE CONSTANT 1010 (TEN) DIRECT (WITHOUT SHIFT)
            TO Y4*/
            PROCEDURE;
            SUBSTR(Y4,9,1)='1'B; SUBSTR(Y4,11,1)='1'B;
            END;
TENL1Y4:    /*SELECT THE CONSTANT 1010 (TEN) LEFT 1 BIT TO Y4.*/
            PROCEDURE;
            SUBSTR(Y4,8,1)='1'B; SUBSTR(Y4,10,1)='1'B;
```

```
        END;
2Y3:    /*SELECT THE CONSTANT 1010 (TEN) LEFT 2 BITS TO Y3*/
        PROCEDURE;
        SUBSTR(Y3,7,1)='1'B; SUBSTR(Y3,9,1)='1'B;
        /*NOTE THAT ALL OTHER Y3 SELECTORS MUST BE OFF*/
        END;
3Y3:    /*SELECT THE CONSTANT 1010 (TEN) LEFT 3 BITS TO Y3*/
        PROCEDURE;
        SUBSTR(Y3,6,1)='1'B; SUBSTR(Y3,8,1)='1'B;
        /*NOTE THAT ALL OTHER Y3 SELECTORS MUST BE OFF*/
        END;
4Y2:    /*SELECT THE CONSTANT 1010 (TEN) LEFT 4 BITS TO Y2*/
        PROCEDURE;
        SUBSTR(Y2,5,1)='1'B; SUBSTR(Y2,7,1)='1'B;
        /*NOTE THAT ALL OTHER Y2 SELECTOR SIGNALS MUST BE OFF*/
        END;
5Y2:    /*SELECT THE CONSTANT 1010 (TEN) LEFT 5 BITS TO Y2*/
        PROCEDURE;
        SUBSTR(Y2,4,1)='1'B; SUBSTR(Y2,6,1)='1'B;
        /*NOTE THAT ALL OTHER Y2 SELECTOR SIGNALS MUST BE OFF */
        END;
6Y1:    /*SELECT THE CONSTANT  1010 LEFT 6 BITS TO Y1.*/
        PROCEDURE;
        SUBSTR(Y1,3,1)='1'B; SUBSTR(Y1,5,1)='1'B;
        /*NOTE THAT ALL OTHER Y1 SELECTOR SIGNALS MUST.BE OFF.*/
        END;
7Y1:    /*SELECT THE CONSTANT 1010 LEFT 7 BITS TO Y1.*/
        PROCEDURE;
        SUBSTR(Y1,2,1)='1'B; SUBSTR(Y1,4,1)='1'B;
        /*NOTE THAT ALL OTHER Y1 SELECTOR SIGNALS MUST BE OFF */
        END;
103:    /*SELECT UH DIRECT (WITHOUT SHIFT) TO Y1, BYTES 0-3*/
        PROCEDURE;
        SUBSTR(Y1,1,32)=SUBSTR(UH,1,32);
        END;
147:    /*SELECT UH DIRECT (WITHOUT SHIFTING) TO Y1, BYTES 4-7*/
        PROCEDURE;
        SUBSTR(Y1,33,32)=SUBSTR(UH,33,32);
        END;
```

2.9 Multiplier Recode and Multiply Extended Precision

2.9.1 General

Multiplication in a digital arithmetic unit is accomplished by over and over additions of the multiplicand as controlled by the multiplier. In its most fundamental form, bits of the multiplier are inspected serially from right to left. If the bit is 1, the multiplicand is added to the partial product and the partial product is then shifted right one bit and back into the accumulator, ready for the next addition. If the bit is zero, the addition is bypassed (or zero is added) and the partial product is again shifted right one bit into the accumulator.

For this class of techniques, a multiplier of length 1 bit requires 1 cycles. A cycle consists of an add, then shift, or merely a shift.

One technique for accelerating the execution of multiplication is to inspect two bits of the multiplier simultaneously, i.e. perform multiplication radix four instead of radix two. In this case, a multiplier of K bits requires only K/2 cycles. Of course, now we must be able to form multiples of 0, 1, 2, and 3 times the multiplicand for addition to the partial product. The 0 and 1 times are easily implemented and the 2 times readily available by a simple left shift. The 3 times multiple, however, is awkward and costly to form. It would probably require an addition of 1 and 2 times the multiplicand.

A useful technique to avoid the necessity for forming the 3 times multiple is to recode the bits of the multiplier being inspected. The inputs to the recoder are bits of the multiplier; the output is a selected multiple of either +2, +1, 0, -1, or -2. All of these multiples may be formed by simple addition or subtraction and possibly a left shift.

Intuitively, we might view this recoding as permitting intentional "mistakes" in forming the multiple at one cycle, but correcting the "mistake" at a subsequent cycle. For example consider a four bit multiplier, MR = 0011. The low-order two bits are $11_2 = 3_{10}$ and thus imply the selection of 3 times the multiplicand. Under the recoding proposed here, however, the bits 11 will select a multiple of -1. On the next cycle a multiple of +1 will be selected, but since the partial product has been shifted 2 bits to the right, this selection actually produces a multiple of +4 relative to the previous cycle. At the end of this second cycle the partial product is correct, i.e. (-1+4)(Multiplicand) = 3 (Multiplicand).

This recoding was taken from DCS Report No. 133, "Suggested Design for a Very Fast Multiplier," by C.S. Wallace.

The recoding actually requires the parallel inspection of three bits of the multiplier. If X_i is the low-order bit of the multiplier, then the bits inspected are X_{i-1}, X_i, and X_{i+1}. The bit X_{i+1} is an extra position at the right of the least significant bit of the multiplier. It is initially 0, but after the first right shift of the multiplier it will equal the previous X_{i-1}, which may not be 0. In a sense, X_{i+1} is the indicator of what "mistake" was made on the previous cycle. The recoding is shown in Table 2.9.1.1. It will accommodate a negative number in two's complement representation.

	X_{i-1}	X_i	X_{i+1}	Multiple Selected
	0	1	1	+2
	0	1	0	+1
or	0	0	1	
	0	0	0	0
or	1	1	1	
	1	1	0	-1
or	1	0	1	
	1	0	0	-2

TABLE 2.9.1.1 Multiple Selected

2.9.2 The Recoding Hardware

An Illiac III AU includes a cascade of four adder/subtractors and four sets of shift gates. One multiplication cycle consists of sequence of four, radix 4 multiplications, i.e. multiplication is perform radix 256. Nine bits of the multiplier stored in the UQ Register are recoded simultaneously to control the shift gates of M Shift Array and the NEG signals of the signed-digit subtracters which determine whether addition or subtraction is performed.

The shifters are all logically identical, however, they are connected to the appropriate SDS so that, with respect to the radix point of the subtracters (between the first and second byte), the values of the multiples are as shown below:

SDS No.	Multiples Selected
1	0, $\pm$128, $\pm$64
2	0, $\pm$32, $\pm$16
3	0, $\pm$8, $\pm$4
4	0, $\pm$2, $\pm$1

The recoding is performed in three bit, overlapping groups according to the specifications in Table 2.9.1.1. Figure 2.9.2.1 illustrates the low-order byte plus the extra right-most bit of the UH register and the shift gates each control. (See Block Diagram, Figure 2.1.1.1).

Notice that the multiples are formed from the largest to the smallest, i.e. $\pm$128 or $\pm$64 are added or subtracted in the first SDS; $\pm$2 and $\pm$1 are added or subtracted in the last SDS. This order is not necessary for multiplication, however, it is a necessity for the execution of division.

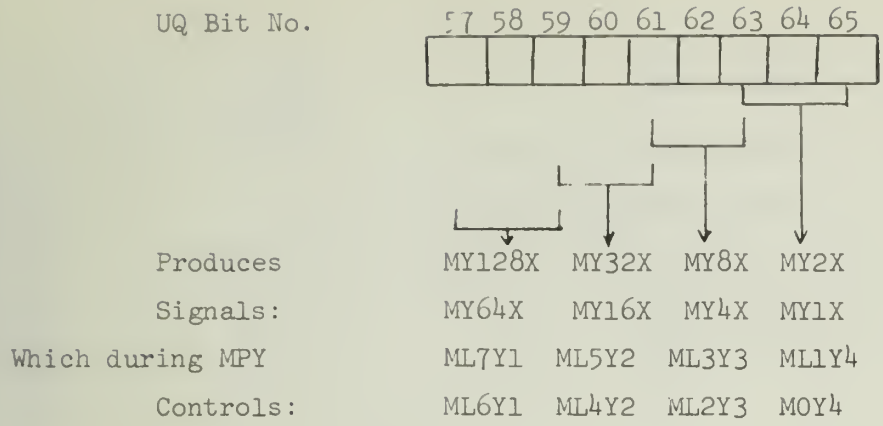


Figure 2.9.2.1 Multiplier Bit, Shift Gate Correspondence

The logic equations actually implemented in Multiplier Recode box are shown below. The Boolean variables SIGN is introduced as a conceptual aid and is not implemented, per se. SIGN = 0 = + ; SIGN = 1 = -. The MYNEG (Multiply Negation) signals are used to set the NEG controls of the SDS to select whether the multiple is added or subtracted.

$$MY128X = (\overline{UQ}_{57}UQ_{58}UQ_{59}) \vee (UQ_{57}\overline{UQ}_{58}\overline{UQ}_{59}$$

$$MY64X = UQ_{58} \oplus UQ_{59}$$

$$SIGN1 = UQ_{57}$$

$$MY32X = (\overline{UQ}_{59}UQ_{60}UQ_{61}) \vee (UQ_{59}\overline{UQ}_{60}\overline{UQ}_{61}$$

$$MY16X = UQ_{60} \oplus UQ_{61}$$

$$SIGN2 = UQ_{59}$$

$$MY8X = (\overline{UQ}_{61}UQ_{62}UQ_{63}) \vee (UQ_{61}\overline{UQ}_{62}\overline{UQ}_{63})$$

$$MY4X = UQ_{62} \oplus UQ_{63}$$

$$SIGN3 = UQ_{61}$$

$$MY2X = (\overline{UQ}_{63}UQ_{64}UQ_{65}) \vee (UQ_{63}\overline{UQ}_{64}\overline{UQ}_{65})$$

$$MY1X = UQ_{64} \oplus UQ_{65}$$

$$SIGN4 = UQ_{63}$$

$$MYNEG0 = \overline{SIGN1} = \overline{UQ}_{57}$$

$$MYNEG1 = SIGN1 \oplus SIGN2 = UQ_{57} \oplus UQ_{59}$$

$$MYNEG2 = SIGN2 \oplus SIGN3 = UQ_{59} \oplus UQ_{61}$$

$$MYNEG3 = SIGN3 \oplus SIGN4 = UQ_{61} \oplus UQ_{63}$$

$$MYNEG4 = \overline{SIGN4} = \overline{UQ}_{63}$$

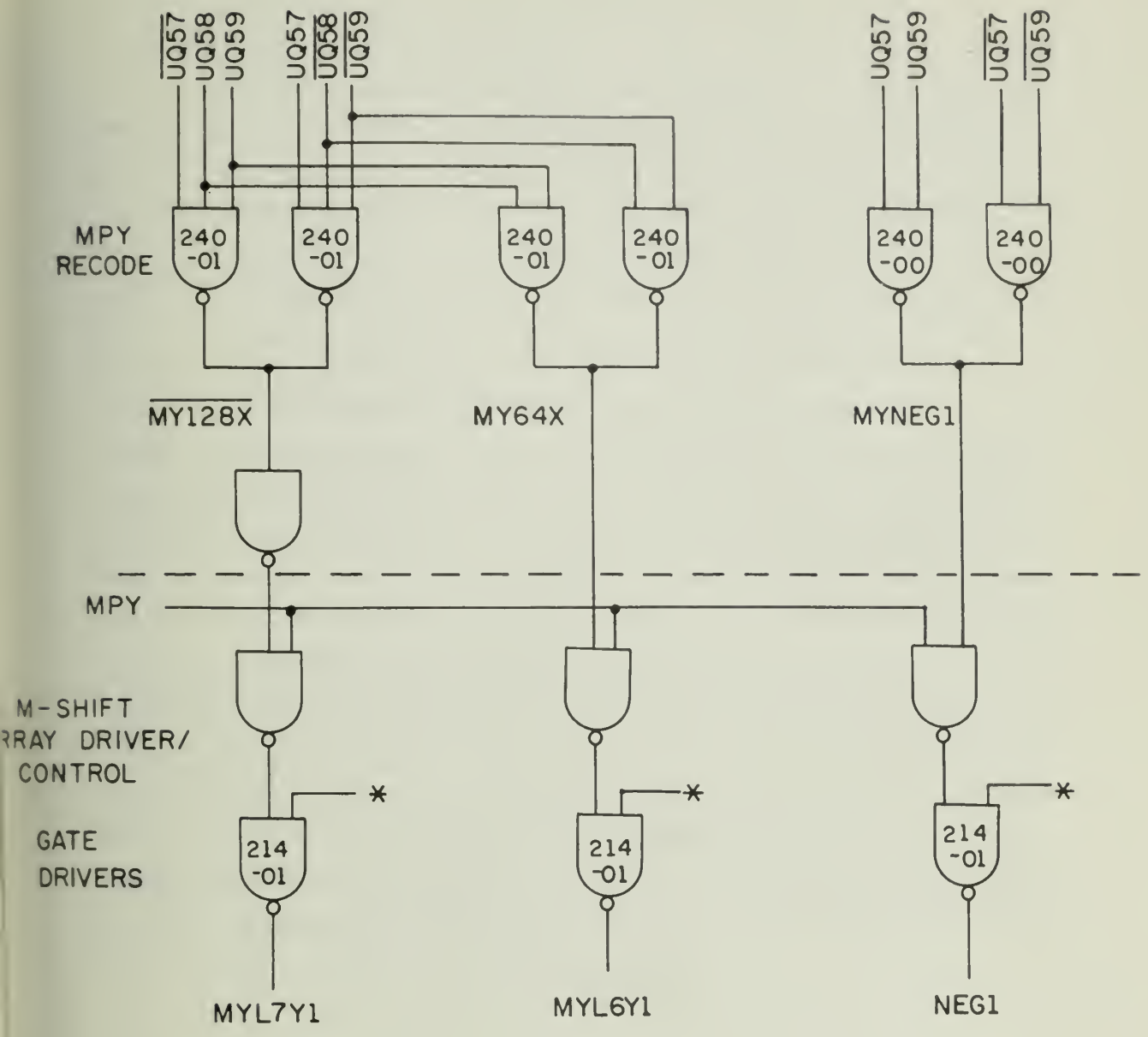
Figure 2.9.2.2 is a typical position of the Multiplier Recode hardware and the connection to the M-Shift Array Driver/Control. UQ_{65} is a special extended position of the UQ register used only in multiplication orders.

Now consider an example of the recoding of the bit pattern shown below:

(all zero) ←	0	0	1	1	1	1	1	0	1	0	0
UQ Position No.	55	56	57	58	59	60	61	62	63	64	65

This pattern is 250_{10} times some appropriate scale factor. Position UQ_{65} is always initially '0' but may have a non-zero value after the contents of UQ are shifted right eight positions. The outputs of the multiplier recode logic would be as follows:

MY128X = 0
MY64X = 0
SIGN1 = 1
MY32X = 0
MY16X = 0
SIGN2 = 1
MY8X = 0
MY4X = 1 } -4 times
SIGN3 = 1
MY2X = 1
MY1X = 0 } -2 times
SIGN4 = 1



* FROM OTHER CONTROL, e.g. DIVISION.

Figure 2.9.2.2 - Typical Position of Multiply Recode and Connection to M-Shift Array Driver Control

After (-4 x multiplicand) + (-2 x multiplicand) is formed, the partial product and the contents of the UQ register are shifted right by eight positions. Now UQ_{57} through UQ_{64} are all zero, but UQ_{65} is '1'. Evaluation of the logic equations shows that only MY1X is selected and that it is to be added to the partial product. The right shift of eight has introduced a factor of 256, thus in this second cycle the addition of one times the multiplicand corresponds to the addition of 256 times on the first cycle. Thus the multiplier 250 is recoded into the form

$$-2 -4 +256 = 250.$$

In floating point multiplication, the multiplier is 7 bytes long. At the end of 7 multiplication cycles, UQ_{65} may be 1 and since UQ_{63} and UQ_{64} are always zero after the seventh cycle (the multiplier for floating point is always positive) an addition multiple of the contents of the M-Register must be added to the partial product in US-UM. This is accomplished during the assimilation cycle.

For long or short integer multiplication the requirement for an extra additon does not arise. If the multiplier is positive, then the high-order bit which is the sign is '0'. Positons UQ_{63}-UQ_{65} will thus be '0' at the end of the last cycle and from the recoding equations it may be seen that no additional multiples are required. If on the other hand the multiplier is negative, UQ_{63}-UQ_{65} will be all ones and again the recoding equations show that no additional multiple is required.

2.9.3 Multiply Extended Precision

The fractional part of the final product is assimilated into the LM-Register and then transferred to the UQ Register for terminal operations and return to the TP. The range of a normalized fraction, f, is given by $1/16 \leq f < 1$. The product of two fractions f_1 and f_2 is thus in the range $1/256 \leq f_1 f_2 < 1$. The fractional part of the product may, therefore, require a terminal left shift of four positions and a corresponding decrease of the exponent. If zeros are injected in the low order four bits when a left shift takes place then the precision of the result is impaired. If, however, the actual four bits of the product are injected, the maximum truncation error is constant for all results from the multiplication operation. These four bits are actually generated and are available in redundant signed-digit form in LM-LS, positions 57 through 60. Without additional hardware, they would be lost by the right shift of 8 positions back to the US-UM Registers. The multiply extended precision hardware assimilates these digits and stores them in case they are needed in the course of the terminal left shift.

Figure 2.9.3.1 is a block diagram of the hardware. The power buffers are merely inverters which keep the propagation logic from unduly loading the outputs of the LS-LM Registers. The propagation logic produces the bits $MEPP_0$ through $MEPP_4$ is illustrated in Figure 2.9.3.2 and defined by the following equations,

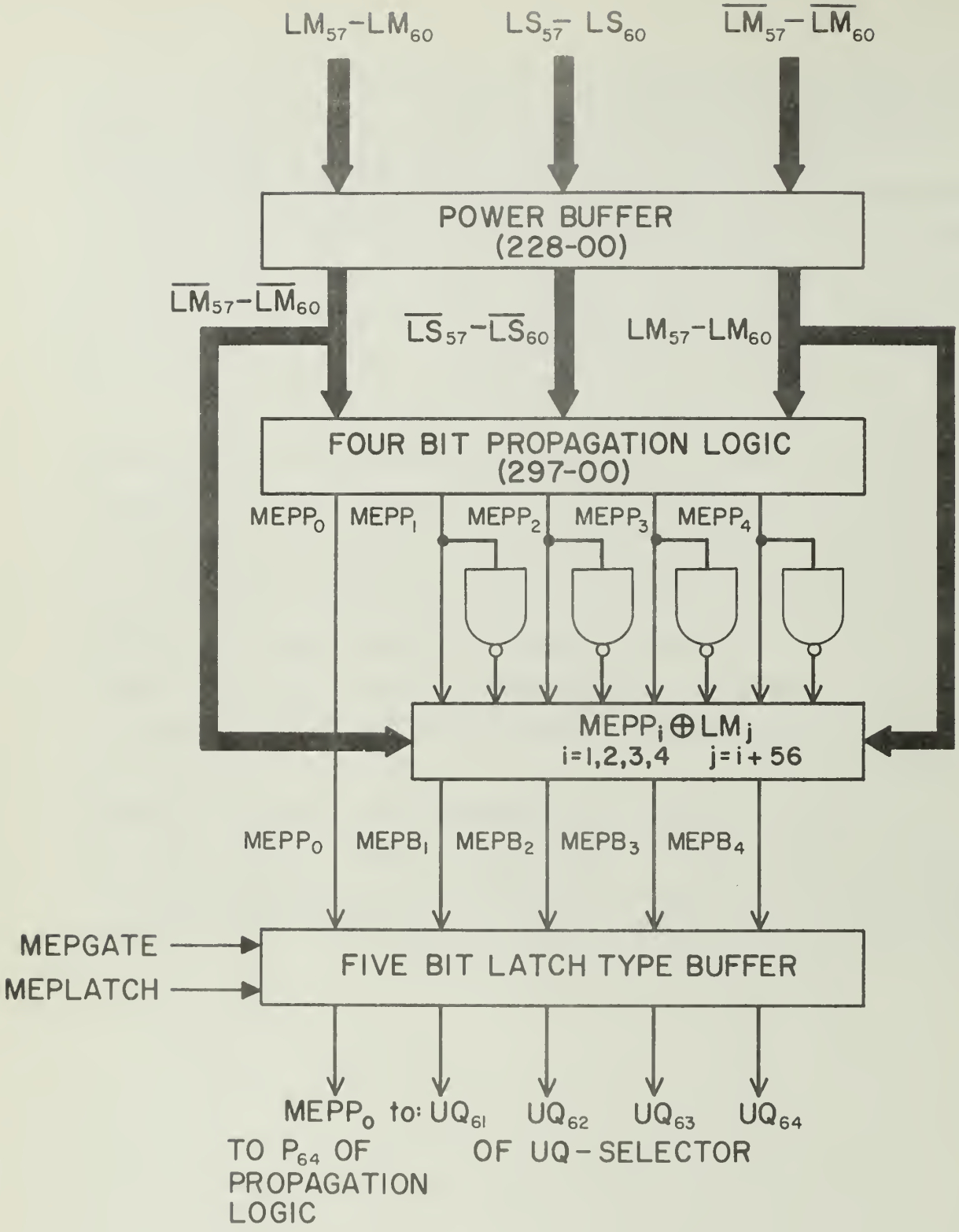


Figure 2.9.3.1 - Block Diagram of Multiply Extended Precision Logic

$$MEPP_4 = \text{'0'}$$

$$MEPP_3 = LS_{60}LM_{60}$$

$$MEPP_2 = LS_{59}LM_{59} \vee \overline{LM}_{59}LS_{60}LM_{60}$$

$$MEPP_1 = LS_{58}LM_{58} \vee \overline{LM}_{58}LS_{59}LM_{59} \vee \overline{LM}_{58}\overline{LM}_{59}LS_{60}LM_{60}$$

$$MEPP_0 = LS_{57}LM_{57} \vee \overline{LM}_{57}LS_{58}LM_{58} \vee \overline{LM}_{57}\overline{LM}_{58}LS_{59}LM_{59} \vee \overline{LM}_{57}\overline{LM}_{58}\overline{LM}_{59}LS_{60}LM_{60}$$

All bits except $MEPP_0$ are combined in an exclusive OR operation with LM_{57} through LM_{60} to produce the assimilated results. $MEPP_0$ is used as the P_{64} input to the full precision propagation logic described in Section 2.7.

Figure 2.9.3.3 is a detail of two positions of the latch type buffer used to store $MEPP_0$ and $MEPB_1$ through $MEPB_4$. The four positions for $MEPB_i$ incorporate the exclusive OR logic used to form LM ⊕ MEPB. The control signal MEPGATE is normally 0, the control signal MEPLATCH is normally 1. When the MEPB bits are to be formed and stored, MEPGATE is set to 1 and the latch, MEPLATCH set down at 0. The latch signal must be returned to 1 prior to the gate, MEPGATE, returning to 0.

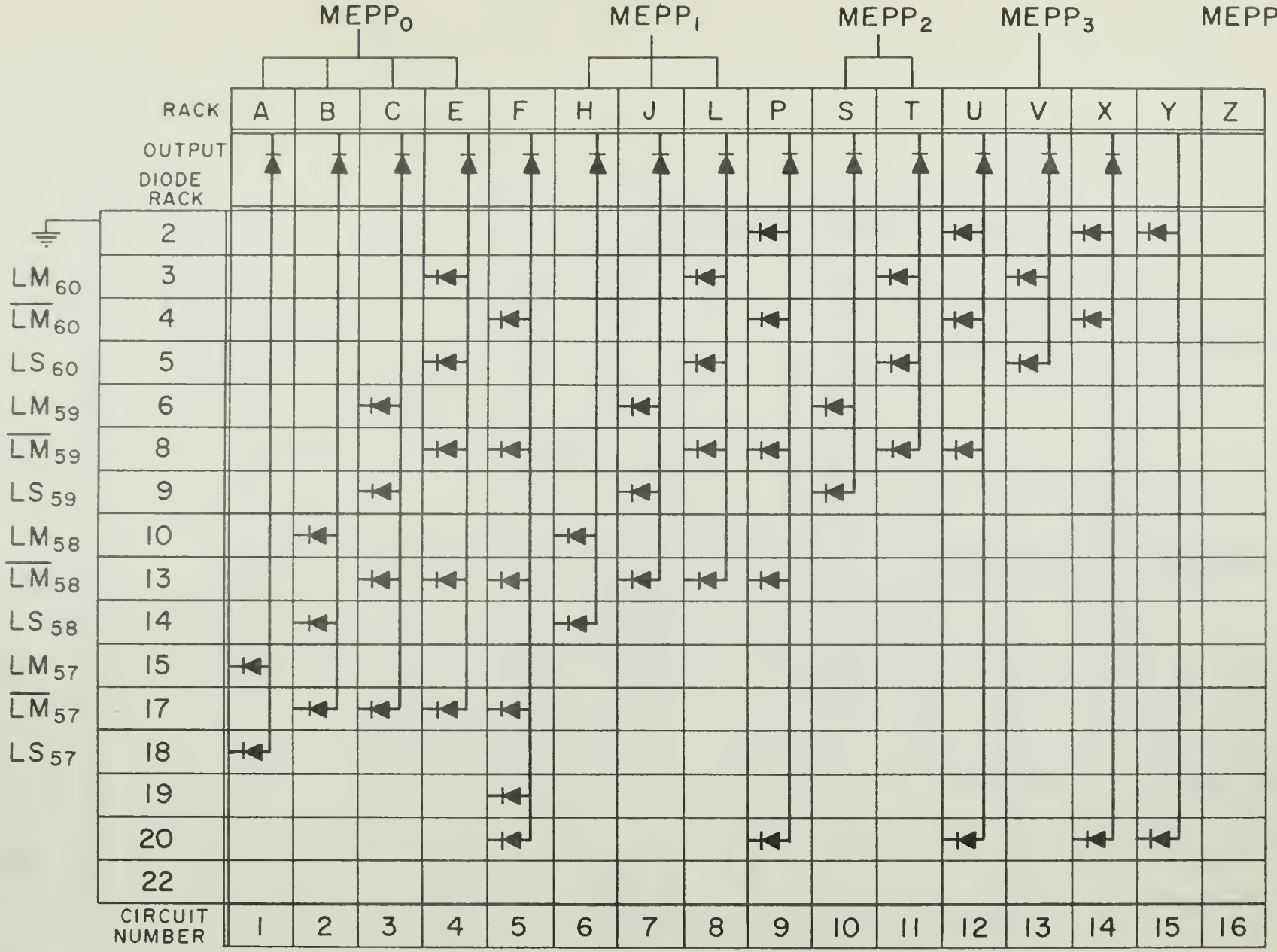


Figure 2.9.3.2 Propagation Logic for Multiply Extended Precision Logic

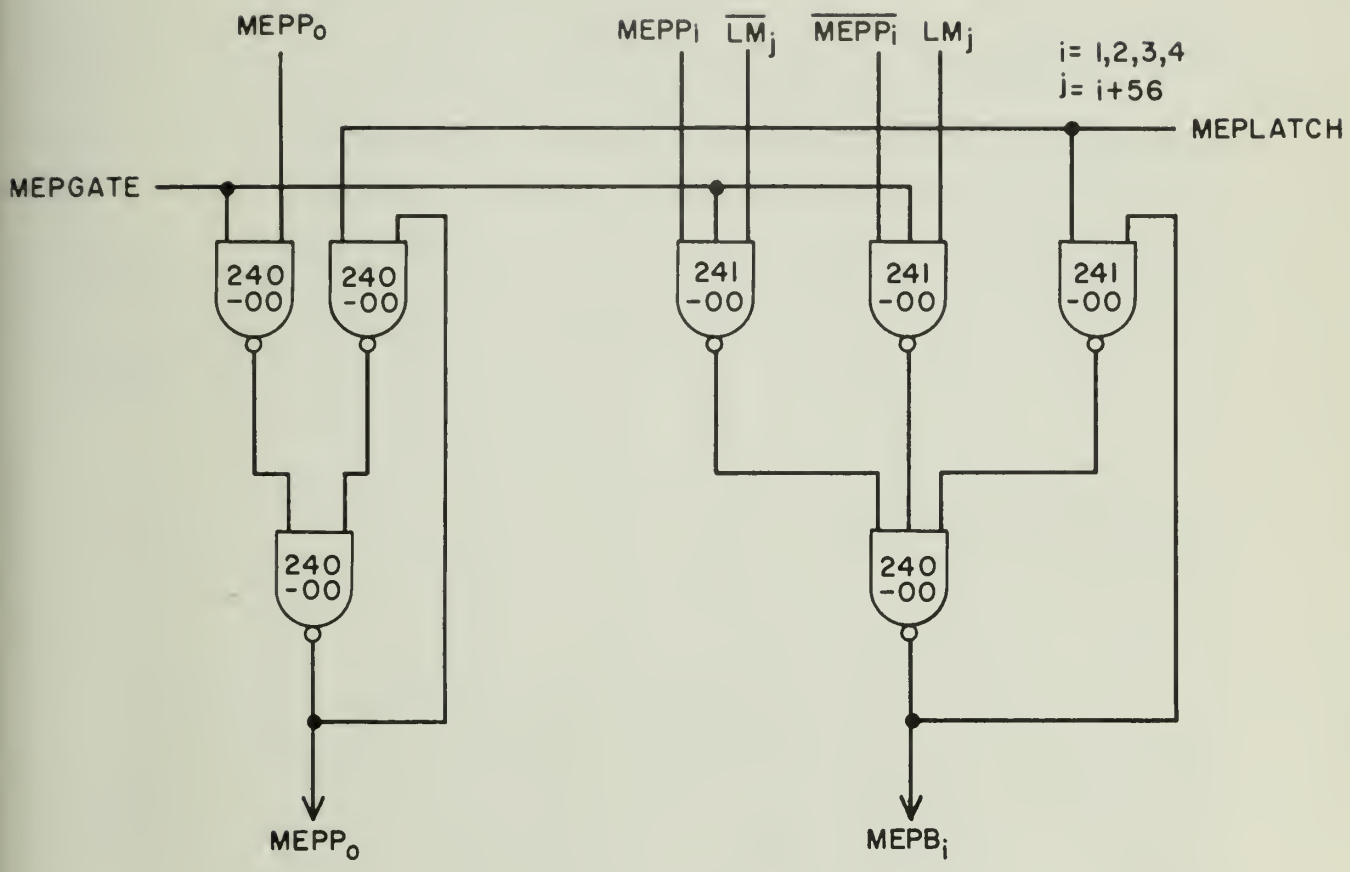


Figure 2.9.3.3 - Exclusive - OR Logic and Latch for Multiply Extended Precision

2.10 Model Division

2.10.1 General

Robertson[1] has proposed a class of division techniques in which quotient digits are selected based upon an inspection of only a few high-order bits of the divisor and partial remainder. The quotient selection mechanism may be viewed as a model of the full precision division mechanism. The model division uses truncated versions of the divisor and partial remainders to produce quotient digits which are in turn used in forming the next full precision partial remainder. The division procedures used in the model need bear no relationship to a conventional division procedure, in particular, to the full precision procedure. The procedure for the model division of Illiac III is a radix 4 table look-up. The nature of this class of division techniques is explored in detail in a paper by Atkins.[2]

The model division determines which multiples of the divisor are to be subtracted from the partial remainder. In this respect it is analogous to the multiplier recoder (Section 2.9) and may, in fact, be viewed as a quotient recoder. In multiplication the recoder introduces redundancy into the representation of the multiplier; in division the recoder introduces redundancy into the representation of the quotient. The quotient recoder is, however complicated by the following properties of the division algorithm:

---

[1] J. E. Robertson, "Methods of selection of quotient digits during digital division," Department of Computer Science, University of Illinois, Urbana, File 663, 1965.

[2] D. E. Atkins, "Higher radix division using estimates of the divisor and partial remainders", IEEE Trans. Computers, vol. C-17, no. 10 (Oct. 1968), pp. 925-934.

1. The quotient recoding is a function of both the divisor and the partial remainder.
2. The partial remainder, unlike the divisor or the multiplier, is not constant throughout the operation.
3. Since partial remainders are formed with the signed-digit subtracters, they are represented redundantly.

But despite these complications, the strong analogy between multiplier recoding and the concept of the model division leads to a division scheme which is highly compatible with the multiplication scheme described in the previous section.

As shown in the Atkins paper, a radix 4 division may be performed using a table look-up on inputs consisting of the four high-order bits of the divisor and the six high-order bits of the shifted partial remainder. The output of the table is a quotient digit value of either $\overline{2}$, $\overline{1}$, 0, 1, or 2. In the most brute force form the table look-up may be thought of as a grid or matrix. The vertical lines are outputs of decoders applied to $\hat{d}$, the truncated (4 bit) version of the divisor; the horizonal lines are outputs of decoders applied to $\hat{rp}$, the truncated (6 bit) version of the shifted partial remainder. At each intersection of the lines is an AND gate with one input connected to the vertical line and the other connected to the horizonal line. Each point of intersection corresponds to a quotient digit value, i, and thus the output of each AND gate is connected to an input of the OR gate with output corresponding to the quotient digit, q=i.

The size of the table is constrained by restrictions on the range of the divisors and partial remainders. The divisor is normalized in the range $1/2 \leq d < 1$. Due to certain properties of the division scheme, any partial remainder, say p_j, must be in the range $|p_j| \leq 2/3\ d$. The shifted partial remainder, rp_j, where r is the radix and j is the recursive index, must be in the range $|rp_j| \leq 8/3\ d$ when $r = 4$. The divisor is always positive; the partial remainder may be either positive or negative since the division is nonrestoring. The actual implementation is not nearly as formidable as the brute force attack might imply.

It is prohibitively expensive to apply the redundant from of the partial remainder directly to a table look-up. In a redundant number system one algebraic value may be disguised in many forms. The 6 digit estimate of the partial remainder is therefore assimilated into a conventional radix complement form prior to the table look-up. The assimilated version is of the form $A_0A_1A_2.A_3A_4A_5A_6$, where A_0 is the sign. The estimates of the divisor are decoded into the intervals shown in Table 2.10.1.1. Since the divisor is stored in the M-Register and since the radix point is between positions 8 and 9, the high-order four bits of the divisor are designated M_9, M_{10}, M_{11} and M_{12}. Note that since d is at least 1/2, M_9 is always 1.

Interval Name	Logic Equations	Range of divisor, d, represented
D_1	$\overline{M}_{10}\overline{M}_{11}\overline{M}_{12}$	$1/2 \leq d < 9/16$
D_2	$\overline{M}_{10}\overline{M}_{11}M_{12}$	$9/16 \leq d < 5/8$
D_3	$\overline{M}_{10}M_{11}\overline{M}_{12}$	$5/8 \leq d < 11/16$
D_4	$\overline{M}_{10}M_{11}M_{12}$	$11/16 \leq d < 3/4$
D_5	$M_{10}\overline{M}_{11}$	$3/4 \leq d < 7/8$
D_6	$M_{10}M_{11}$	$7/8 \leq d < 1$
D_7	$D_1 \vee D_2 \vee D_3$	$1/2 \leq d < 11/16$
D_8	$D_4 \vee D_5 \vee D_6$	$11/16 \leq d < 1$
D_9	$D_4 \vee D_5$	$11/16 \leq d < 7/8$
D_{10}	$(D_5 \vee D_6) = M_{11}$	$3/4 \leq d < 1$
D_{11}	$(D_1 \vee D_2 \vee D_3 \vee D_4) = \overline{M}_{11}$	$1/2 \leq d < 3/4$

Table 2.10.1.1 - Divisor Interval Selection Logic

The assimilated estimate of the partial remainder and the outputs of the divisor interval selection logic are used to generate the logic signals ZERO, ONE, and TWO corresponding to quotient digit magnitudes of 0, 1, and 2, respectively. The signals ZERO and TWO are formed as the OR of two other signals, one corresponding to the quadrant for positive partial remainders; the other corresponding to negative partial remainders. The signal, ONE, is implemented in the form ONE= $\overline{\text{ZERO v TWO}}$. The following defines ZERO and TWO:

$$\text{ZERO} = \text{ZEROP v ZERON}$$

$$\text{TWO} = \text{TWOP v TWON}$$

$$\text{ZEROP} = \bar{A}_0\bar{A}_1\bar{A}_2\bar{A}_3\bar{A}_4 \text{ v } \bar{A}_1\bar{A}_2\bar{A}_3\bar{A}_4\bar{A}_5\bar{D}_1 \text{ v } \bar{A}_0\bar{A}_1\bar{A}_2\bar{A}_3D_{10}$$

$$\text{ZERON} = A_0A_1A_2A_3A_4 \text{ v } A_0A_1A_2A_3\bar{A}_4A_5D_{10}$$

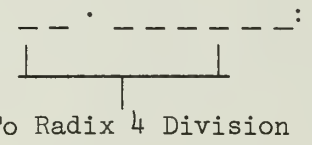
$$\begin{aligned}\text{TWOP} = {} & \bar{A}_0A_3A_4\bar{A}_5A_6D_1 \text{ v } \bar{A}_0A_3A_4A_5D_1 \text{ v } \bar{A}_0A_3A_4A_5A_6D_2 \\ & \text{v } \bar{A}_0A_2A_3D_8 \text{ v } \bar{A}_0A_2\bar{A}_3A_4D_9 \text{ v } \bar{A}_0A_2\bar{A}_3\bar{A}_4A_5D_4 \\ & \text{v } \bar{A}_0A_2\bar{A}_3A_4A_5A_6D_6 \text{ v } \bar{A}_0A_1 \text{ v } \bar{A}_0A_2D_7\end{aligned}$$

$$\begin{aligned}\text{TWON} = {} & A_0\bar{A}_3\bar{A}_4D_1 \text{ v } A_0\bar{A}_3\bar{A}_4\bar{A}_5D_2 \text{ v } A_0\bar{A}_3\bar{A}_4\bar{A}_5\bar{A}_6D_3 \\ & \text{v } A_0\bar{A}_2A_3A_4\bar{A}_5\bar{A}_6D_5 \text{ v } A_0\bar{A}_2A_3\bar{A}_4\bar{A}_5D_6 \text{ v } A_0\bar{A}_2D_{11} \\ & \text{v } A_0\bar{A}_2A_3\bar{A}_4D_5 \text{ v } A_0\bar{A}_2\bar{A}_3 \text{ v } A_0\bar{A}_1\end{aligned}$$

2.10.2 Functional Description

We have now defined a radix 4 quotient selection mechanism which is analogous to the multiplier recoder defined in Table 3. As with multiplication, division is extended to radix 256 by means of four successive applications of radix 4 division.

Let the figure below represent the high-order byte of the US-UM Register and let : denote the radix point for the full precision division.



A radix 4 division with radix point denoted '.' is applied to the leading 6 positions of the output of US-UM. For a radix 256 division the magnitude of the shifted partial remainder is less than 170 2/3 relative to the radix point,':'. The shifted partial remainder relative to '.' is therefore less than 8/3. The radix 4 table look-up selects a quotient digit magnitude of either 0,1, or 2. These correspond to radix 256 digits of 0, 64, or 128 and to the selection of no shift gates, shift gate ML7Y1, or ML6Y1. If neither gate is selected the Y input to the subtracter is zero. The selected multiple of the divisor is added in the first signed-digit subtracter (SDS1 in Figure 2.1.1.1) if the sign of the partial remainder, A_0, is 1; it is subtracted if A_0 is 0. The new partial remainder, the next input to the model division, appears at the output of SDS1. For the next radix 4 division, rather than shifting the partial remainder left two positions, the input to the model is shifted right by two positions. Figure 2.10.2.1 summarizes all four stages of one pass through the subtracter cascade.

Position Number	Shift Gate Selected for magnitude of quotient digit =		
1 2 3 4 5 6 7 8 9 10 11 12	2	1	0
Output of US-UM _ _ . _ _ _ _ _ _ : _ _ _ _ (To Model)	ML7Y1	ML6Y1	none
Output of SDS 1 0 0 _ _ . _ _ _ _ : _ _ _ _ (To Model)	ML5Y2	ML4Y2	none
Output of SDS 2 0 0 0 0 _ _ . _ _ : _ _ _ _ (To Model)	M13Y3	ML2Y3	none
Output of SDS 3 0 0 0 0 0 0 _ _ : _ _ _ _ (To Model)	ML1Y4	MDY4	none

Note: The symbol . represents the radix point for the radix 4 model division. The symbol : represents the radix point for the full precision division, radix 256.

SETTING OF NEG SIGNALS:

Division Stage No.	Positive Partial Remainder ($A_o = 0$)	Negative Partial Remainder ($A_o = 1$)
1	NEG0 = NEG1 = 1	NEG0 = NEG1 = 0
2	NEG1 = NEG3 = 1	NEG1 = NEG2 = 0
3	NEG2 = NEG3 = 1	NEG2 = NEG3 = 0
4	NEG3 = NEG4 = 1	NEG3 = NEG4 = 0

Figure 2.10.2.1 - Connection of Model Division to Full Precision Structure

Figure 2.10.2.2 is a block diagram of the entire model division structure. The Input Gating is an AND-OR complex which under control of signals C_1 through C_4 gates the appropriate digits of successive partial remainders into the Assimilation box. Before continuing with the description we must note a slight complication again arising from bogus overflow. In Figure 2.10.2.1, as the inputs to the model division are moved to the right, zeros are shown occupying all positions to the left of the highest order input. The range restrictions on the shifted partial remainders are such that the positions shown as zero should indeed be zero if the partial remainders were not in a redundant form. But due to bogus overflow, the highest order digit of the input to the model may be 1 with a $\overline{1}$ in the position immediately to the left, or vice-versa. To compensate for this behavior the magnitude of the digit immediately to the left of the model input is monitored. If it is non-zero, then the sign of the high-order digit into the model is complemented as it is gated into the Assimilation box. Note that the 0th bit of the UM-Register is equivalent to the 8th position of the LM-Register.

The Assimilation box produces a two's complement version of the estimate of the shifted partial remainder. This together with the Division Interval Select Logic drives the Quotient Select Table. The quotient digits are represented in the same signed digit format as produced by the subtracters. The following gives the signed digit representation of each quotient digit value:

Quotient Digit	Representation
+2	1 0
+1	0 1
0	0 0
$\overline{0}$	$\overline{0}$ $\overline{0}$
$\overline{1}$	$\overline{0}$ 1
$\overline{2}$	$\overline{1}$ 0

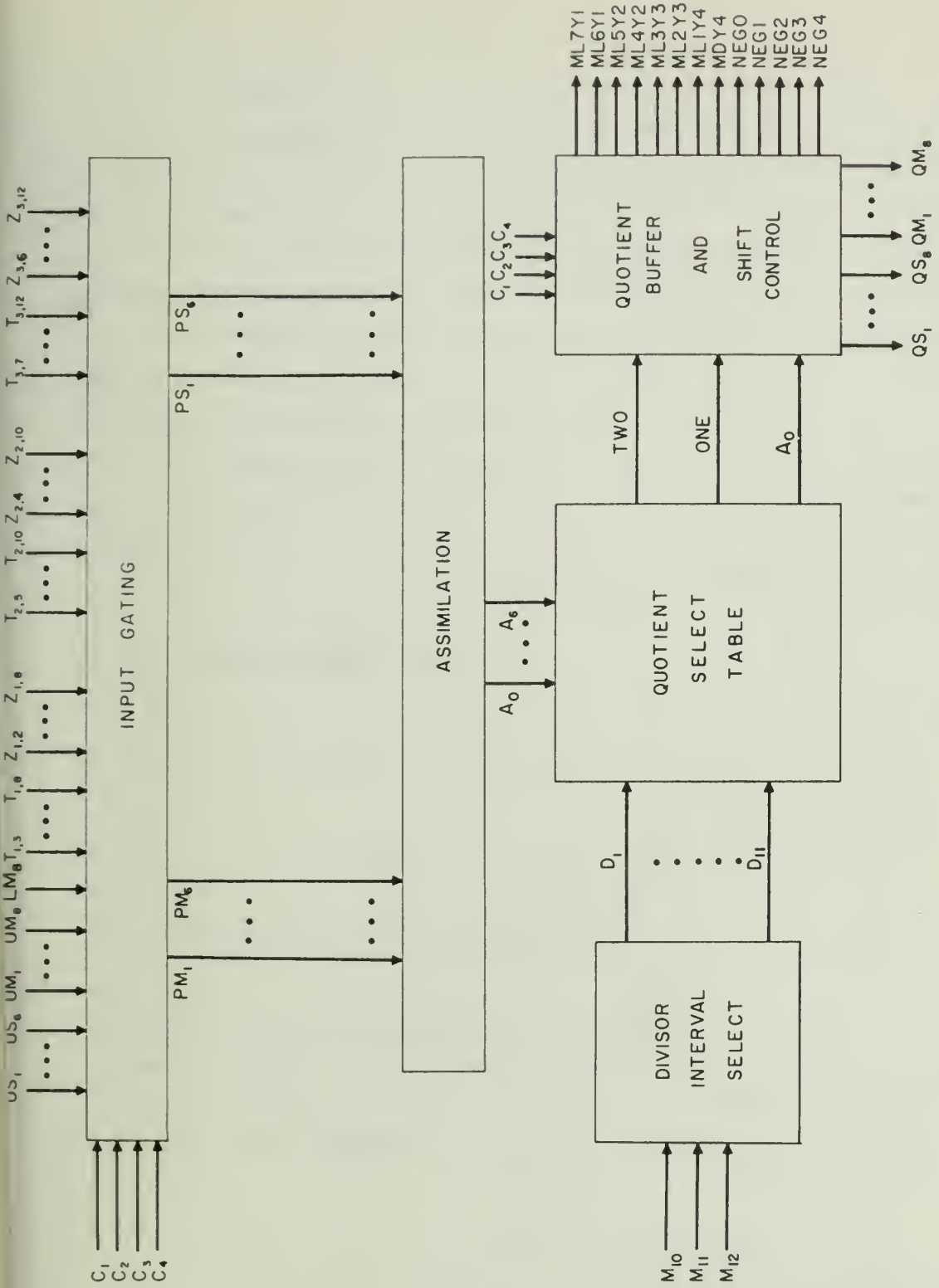


Figure 2.10.2.2 - Block Diagram of Model Division

Note that a distinction is made between a positive and negative zero. The sign of all digits, including zero, is the same as the sign of the partial remainder. If the digit 0 is formed then zero is subtracted from the partial remainder. If the digit $\overline{0}$ is formed then zero is added to the partial remainder. As shown in the proof in the Appendix of DCL Report No. 333 this method of handling a zero quotient digit eliminates bogus overflow at position 1 for division.

The quotient digits are buffered until eight are collected. They are then gated to the low-order byte of the UH-UQ Register. The quotient digits also setup the shift gates and NEG signal in accordance with description in Figure 2.10.2.2. The operating time of the model is summarized in Table 2.10.2.1.

Block	No. of Collector Delays
Input Gating	2
Assimilation	3
Quotient Selection	2
Quotient Storage and Shift Control	3
Total	10

Table 2.10.2.1 - Operating Times of the Model Division

It should be emphasized that the scheme used in the model division is but one of many possibilities. Since the amount of logic involved is quite small (10 cards), and has a well defined interface and is physically one package, it is quite feasible to replace the model with new, hopefully improved versions.

2.10.3 Implementation

The input gating (Figure 2.10.2.2) is essentially a 4-way selector which permits the truncated version of the partial remainder (6 high-order digits) to be gated to the model division from one of four stations. Each position of the selector is formed by connecting together the collectors of four, 228 NAND gates. One of the gates is a 228-03 variant with the normal 2.2K ohm collector resistor; the other three are 228-14 variants with 12K ohm collector resistors. Drawing No. 210-01 details the input gating logic.

The selected estimate of the partial remainder is gated onto the PM and PS lines. At this point the partial remainder is in signed-digit format. PM are magnitude bits and PS are the corresponding sign bits. The high-order position PS_1, PM_1 must be subjected to the same bogus overflow correction remedy as described in Section 2.5.2. The signals D0DM0, D1DMD, D2DMD, and D3DMD are the gate signals. They and the bogus overflow logic are defined in the PL/1 description in Section 2.10.4. Note that D0DMD through D3DMD are equivalent to the signals C1 through C4 shown on the block diagram, Figure 2.10.2.2.

The output of the input gating logic, PS and PM, drive the assimilation logic. Here the estimate of the partial remainder is converted to a conventional, 2's complement form. The logic used is the same as that described in conjunction with the propagation logic (Section 2.7). First borrow bits, B, are generated by a lookahead scheme. These bits are defined as follows:

$$B_6 = 0$$

$$B_5 = PS_6PM_6$$

$$B_4 = PS_5PM_5 \vee \overline{PM_5}\ PS_6\ PM_6$$

$$B_3 = PS_4PM_4 \vee PM_4PS_5PM_5 \vee \overline{PM_4}PM_5PS_6PM_6$$

$$B_2 = PS_3PM_3 \vee \overline{PM_3}PS_4PM_4 \vee \overline{PM_3}\,\overline{PM_4}PS_5PM_5 \vee \overline{PM_3}\,\overline{PM_4}\,\overline{PM_5}PS_6\ PM_6$$

$$B_1 = PS_2PM_2 \vee \overline{PM_2}PS_3PM_3 \vee \overline{PM_2}\ \overline{PM_3}PS_4PM_4$$
$$\vee\ \overline{PM_2}\,\overline{PM_3}\,\overline{PM_4}PS_5PM_5 \vee \overline{PM_2}\,\overline{PM_3}\,\overline{PM_4}\,\overline{PM_5}PS_6PM_6$$

These B-bits are produced by 297-03 diode-matrix board shown

in Figure 2.10.3.1 and on Drawing No. 210-02. The B-bits are combined in an exclusive-OR with the PM-bits to produce the A-bits. A_o through A_6 are the bits of the 2's complement form of the partial remainder. Specifically:

$$A_i = B_i \oplus PM_i \quad \text{for } i = 1, 6$$

Since $B_6 = 0$, $A_6 = PM_6$.

A_o is the sign bit and is identical to B_o.

$$A_o = B_o = (PS_1\ PM_1 \vee \overline{PM_1}\ B_1) \cdot \overline{ZERO}$$

The ANDing with $\overline{ZERO}$ guarantees that zero will always be assigned a positive sign ($A_o = 0$). B_o is generated by logic shown on Drawing No. 210-04.

The divisor interval selection logic is also illustrated on Drawing No. 210-02. The logic converts the high order 4 bits of divisor into interval signals which are used by the quotient selection table. Note that due to normalization the high order bit of the divisor, M_9, is always '1' and is therefore not explicitly used in the divisor interval logic. The logic equations implemented in this logic are defined in Table 2.10.1.1. Figure 2.10.3.2 is a detail of the logic used to implement these equations.

The signal MDDINT may be thought of as a gate on the input to the divisor interval selection logic. It will be set to 1 for floating and fixed point division. The model division hardware is, however also used for conversion from binary to decimal. The conversion algorithm is based upon integer binary division by $1010_2 = 10_{10}$. In this case MDDINT is left at 0, and the signal DRTEN is set to 1. This signal causes the divisor to appear to be the constant 1010; the divisor is not explicitly stored in the M-Register.

The assimilated version of the high-order bits of the divisor, A_o through A_6, and the output of the divisor interval selection logic drive the quotient selection diode matrix. This logic implements the equations given in Section 2.10.1 - 4/4. Figure 2.10.3.3 illustrates the table upon which the selection for positive partial remainders is based. Figure 2.10.3.4 illustrates the table upon which the selection

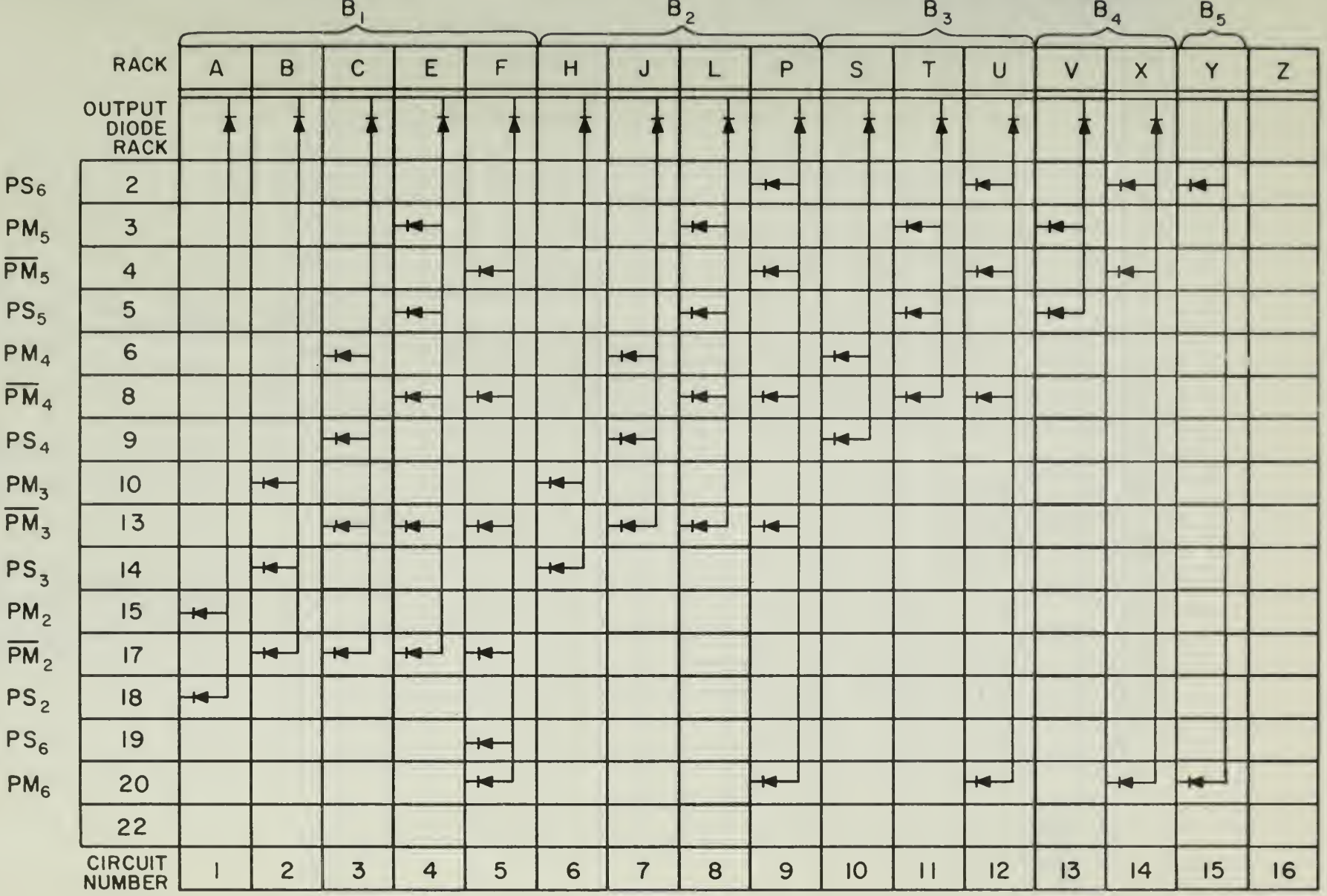
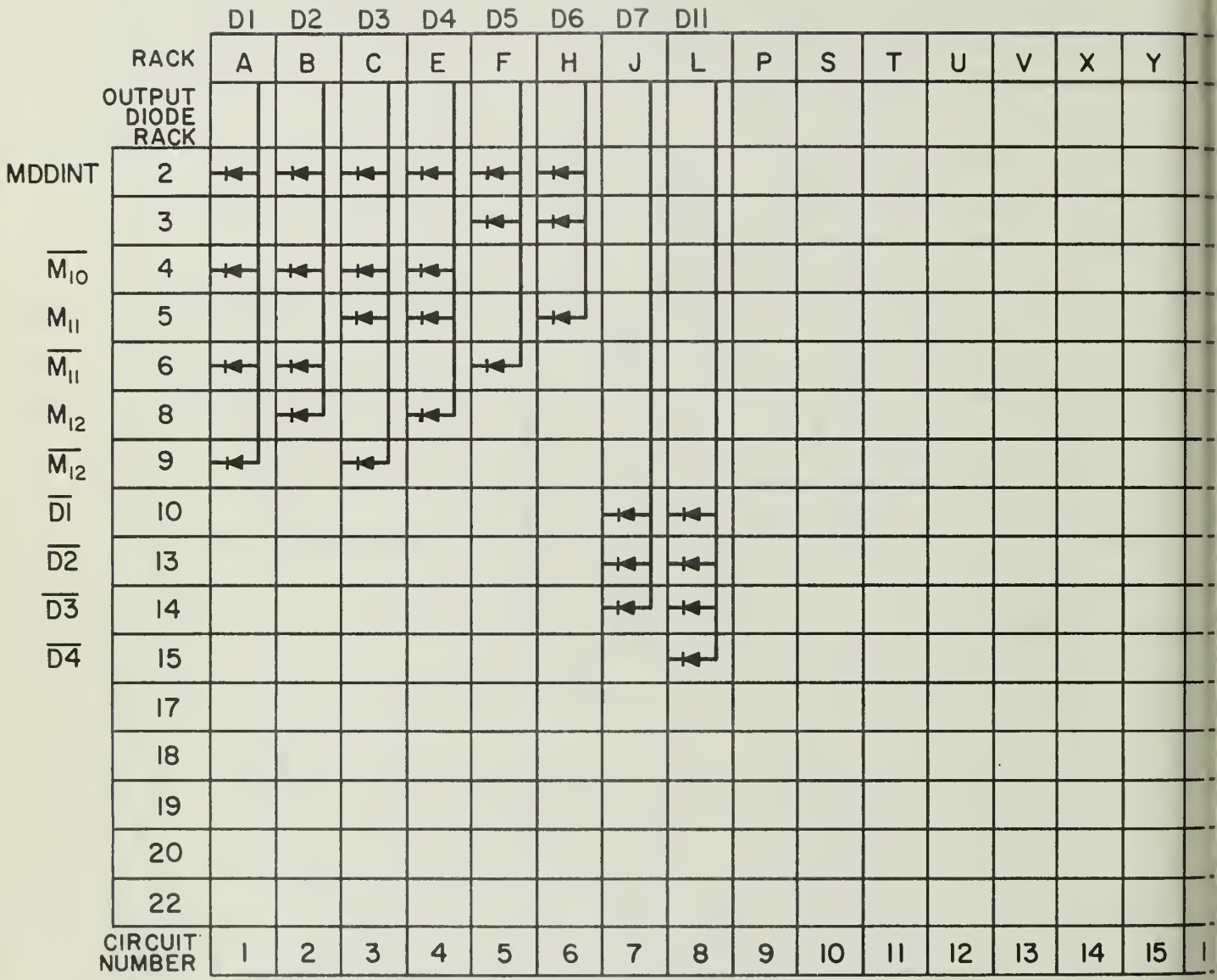


Figure 2.10.3.1 Propagation Diode Matrix



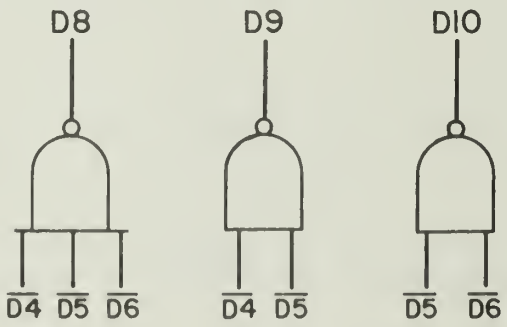


Figure 2.10.3.2 Divisor Interval Select Logic

A Bit No. 012 3456								
010.1100	2	2	2	2	2	2	2	2
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
001.1100	2	2	2	2	2	2	2	2
001.1011	2	2	2	2	2	2	2	2
001.1010	2	2	2	2	2	2	2	2
001.1001	2	2	2	2	2	2	2	2
001.1000	2	2	2	2	2	2	2	2
001.0111	2	2	2	2	2	2	2	2
001.0110	2	2	2	2	2	2	1	1
001.0101	2	2	2	2	2	2	1	1
001.0100	2	2	2	2	2	2	1	1
001.0011	2	2	2	2	1	1	1	1
001.0010	2	2	2	2	1	1	1	1
001.0001	2	2	2	1	1	1	1	1
001.0000	2	2	2	1	1	1	1	1
000.1111	2	2	1	1	1	1	1	1
000.1110	2	1	1	1	1	1	1	1
000.1101	2	1	1	1	1	1	1	1
000.1100	1	1	1	1	1	1	1	1
000.1011	1	1	1	1	1	1	1	1
000.1010	1	1	1	1	1	1	1	1
000.1001	1	1	1	1	1	1	1	1
000.1000	1	1	1	1	1	1	1	1
000.0111	1	1	1	1	0	0	0	0
000.0110	1	1	1	1	0	0	0	0
000.0101	1	0	0	0	0	0	0	0
000.0100	1	0	0	0	0	0	0	0
000.0011	0	0	0	0	0	0	0	0
000.0010	0	0	0	0	0	0	0	0
000.0001	0	0	0	0	0	0	0	0
000.0000	0	0	0	0	0	0	0	0
Divisor $\hat{d}$	.1000	.1001	.1010	.1011	.1100	.1101	.1110	.1111

Figure 2.10.3.3 - Quotient Selection for Positive Partial Remainders

A Bit No. 0 1 2 3 4 5 6	$\hat{d}$ = .1000	.1001	.1010	.1011	.1100	.1101	.1110	.1111
1 1 1.1 1 1 1	0	0	0	0	0	0	0	0
1 1 1.1 1 1 0	0	0	0	0	0	0	0	0
1 1 1.1 1 0 1	0	0	0	0	0	0	0	0
1 1 1.1 1 0 0	0	0	0	0	0	0	0	0
1 1 1.1 0 1 1	1	1	1	1	0	0	0	0
1 1 1.1 0 1 0	1	1	1	1	0	0	0	0
1 1 1.1 0 0 1	1	1	1	1	1	1	1	1
1 1 1.1 0 0 0	1	1	1	1	1	1	1	1
1 1 1.0 1 1 1	1	1	1	1	1	1	1	1
1 1 1.0 1 1 0	1	1	1	1	1	1	1	1
1 1 1.0 1 0 1	1	1	1	1	1	1	1	1
1 1 1.0 1 0 0	1	1	1	1	1	1	1	1
1 1 1.0 0 1 1	2	1	1	1	1	1	1	1
1 1 1.0 0 1 0	2	1	1	1	1	1	1	1
1 1 1.0 0 0 1	2	2	1	1	1	1	1	1
1 1 1.0 0 0 0	2	2	2	1	1	1	1	1
1 1 0.1 1 1 1	2	2	2	2	1	1	1	1
1 1 0.1 1 1 0	2	2	2	2	1	1	1	1
1 1 0.1 1 0 1	2	2	2	2	1	1	1	1
1 1 0.1 1 0 0	2	2	2	2	2	2	1	1
1 1 0.1 0 1 1	2	2	2	2	2	2	1	1
1 1 0.1 0 1 0	2	2	2	2	2	2	1	1
1 1 0.1 0 0 1	2	2	2	2	2	2	2	2
1 1 0.1 0 0 0	2	2	2	2	2	2	2	2
1 1 0.0 1 1 1	2	2	2	2	2	2	2	2
1 1 0.0 1 1 0	2	2	2	2	2	2	2	2
1 1 0.0 1 0 1	2	2	2	2	2	2	2	2
1 1 0.0 1 0 0	2	2	2	2	2	2	2	2
1 1 0.0 0 1 1	2	2	2	2	2	2	2	2
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
1 0 1.0 1 0 0	2	2	2	2	2	2	2	2

Figure 2.10.3.4 - Quotient Selection for Negative Partial Remainders

of the negative partial remainders is based. Figures 2.10.3.5 and 2.10.3.6 illustrate the actual logic with which the tables are implemented.

At this point refer back to the block diagram, Figure 2.10.2.2. Note that the output of the Quotient Selection Table drives the quotient buffer and shift control. The divisor algorithm for the main subtracter cascade (Upper to Lower Registers) is radix 256, i.e. 8 quotient digits are formed. This is accomplished by four applications on the radix 4 table look-up scheme. The results of each of these radix four divisions (two signed digits) are stored in the quotient buffer until eight digits have been formed. At that point the eight digits are transferred to the low-order byte of the UH - UQ Registers. The digits stored in the quotient buffer also activate the M-Shift gates in order to form the next partial remainder. Detailed logic for the quotient buffer is given on Drawing Nos. 210-04 and 210-05.

The quotient buffer consists of two 8 bit registers, the QM-Register and QS-Register. The positions of the register are designated 1 through 8, left to right. The buffer is loaded in four steps under control of the signals LDQMS12, LDQMS34, LDQMS56, and LDQMS78. The quotient is represented in signed-digit format. QM_i holds the magnitude bit and QS_i the corresponding sign. The loading of the positions of the QM and QS Registers is defined as follows: (Note that QM and QS are initially clear.)

$$TWO = TWOP \vee TWON$$

$$ZERO = ZEROP \vee ZERON$$

$$\text{ØNE} = \overline{TWO} \cdot \overline{ZERO}$$

$$QM_1 = TWO \cdot LDQMS12$$

$$QM_2 = ONE \cdot LDQMS12$$

$$QS_1 = QS_2 = A_o \cdot LDQMS12$$

$$QM_3 = TWO \cdot LDQMS34$$

$$QM_4 = \text{ØNE} \cdot LDQMS34$$

$$QS_3 = QS_4 = A_o \cdot LDQMS34$$

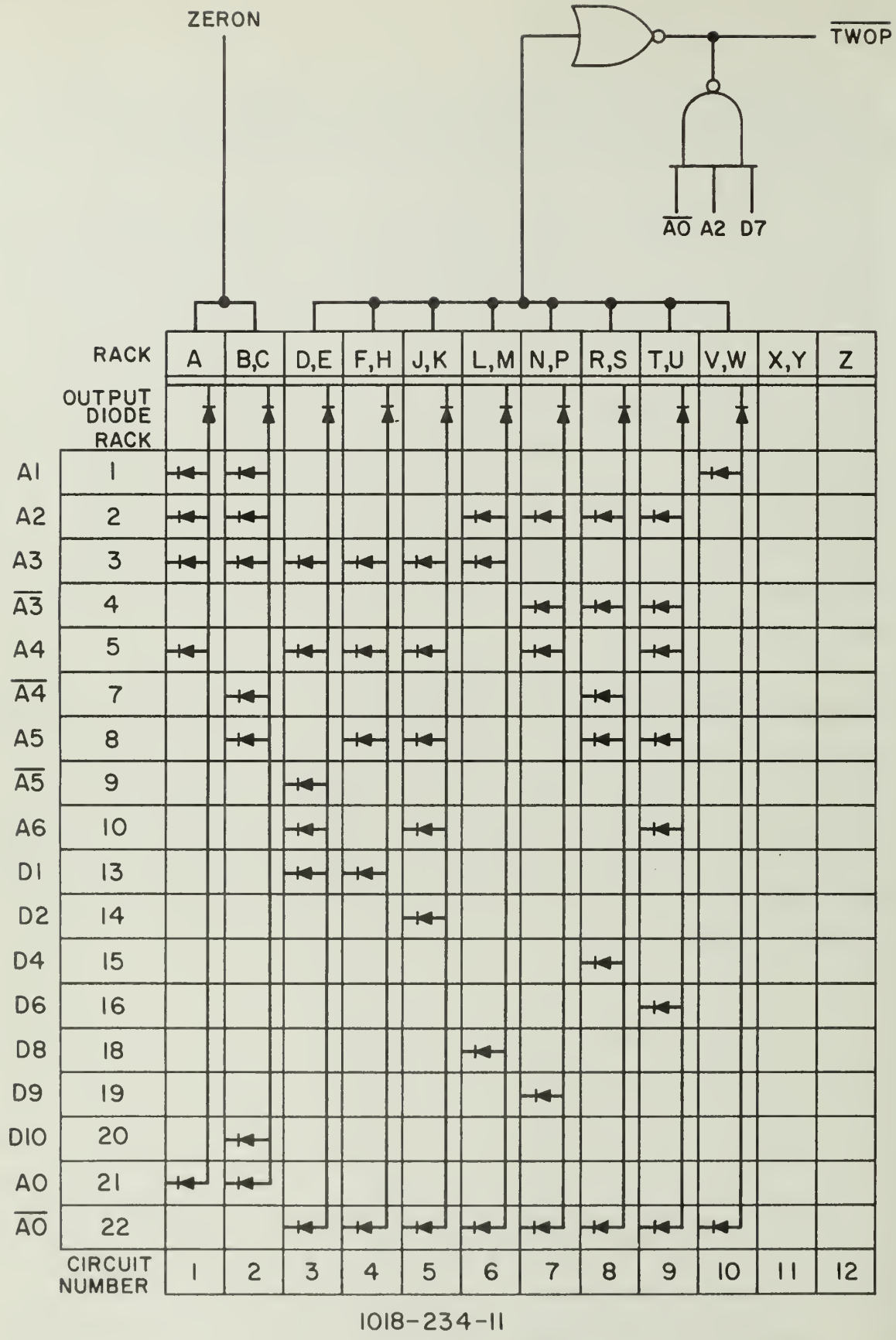


Figure 2.10.3.5 Quotient Selection Logic for ZERON and TWOP

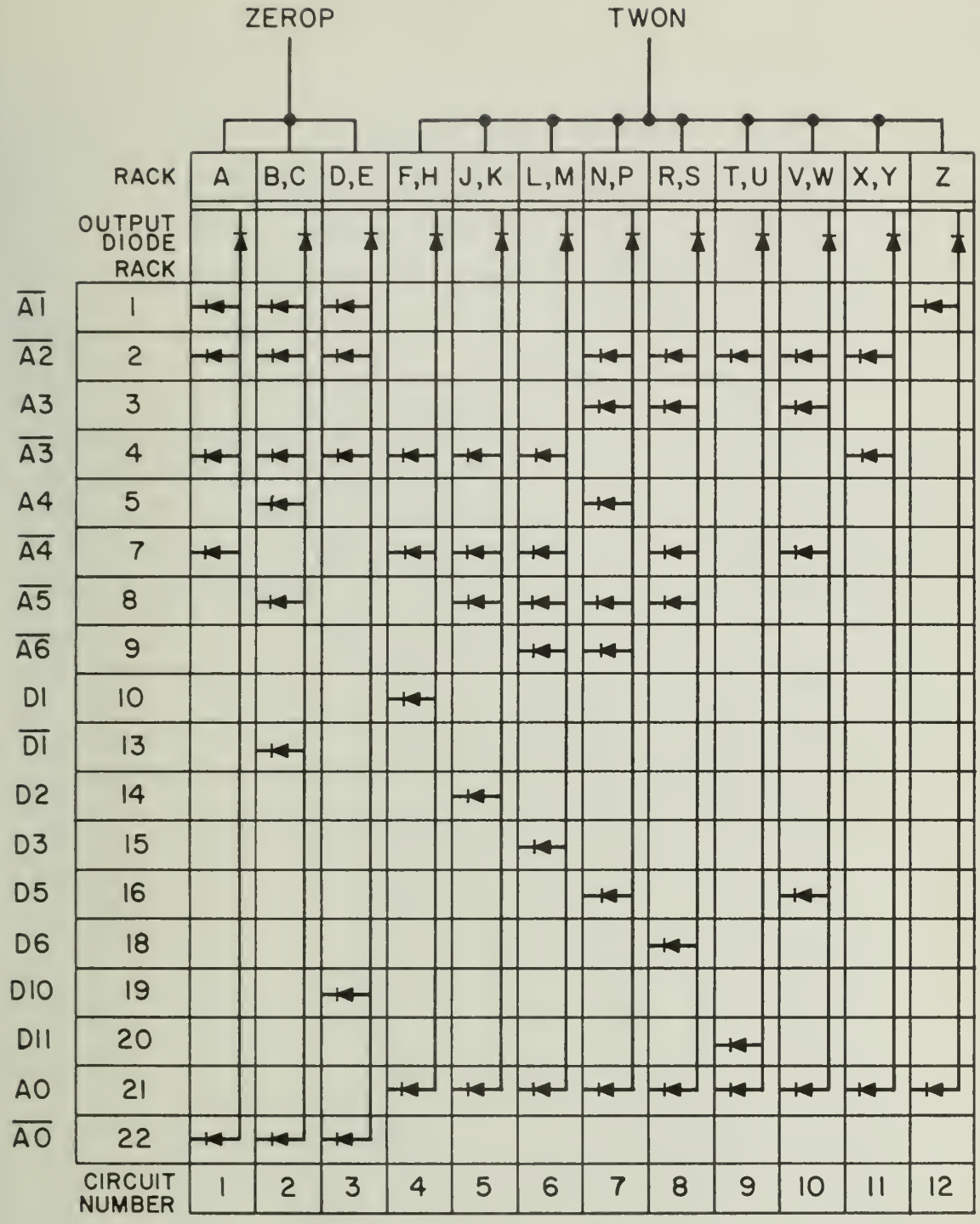


Figure 2.10.3.6 - Quotient Selection Logic for ZEROP and TWON

QM_5 = TWO LDQMS56

QM_6 = ONE LDQMS56

$QS_5 = QS_6 = A_o$ LDQMS56

QM_7 = TWO L DQMS78

QM_8 = ØNE LDQMS78

$QS_7 = QS_8 = A_o$ LDQMS78

The true outputs of the QM Register are coupled through gates (228-03) to the "DV" inputs of the drivers for the M-Shift Array as shown in Figure 2.8.2.2. The correspondence between the QM signals, the DV signals, and the M-Shift Signals is defined below. DIV is a gate signal which enables the M-Shift drivers to be controlled by the model division rather than the Multiplier Recorder.

ML7Y1 = DV128X = DIV QM_1

ML6Y1 = DV64X = DIV QM_2

ML5Y2 = DV32X = DIV QM_3

ML4Y2 = DV16X = DIV QM_4

ML3Y3 = DV8X = DIV QM_5

ML2Y3 = DV4X = DIV QM_6

ML1Y4 = DV2X = DIV QM_7

MDY4 = DV1X = DIV QM_8

The setting of the NEG signals into the signed-digit subtracters (Section 2.5) determine whether the selected multiple of the divisor is combined by addition or subtraction with the partial remainder. The NEG signals, outputs of flip-flops, are set as a function of the sign of the quotient selected and as a function of which radix four division sub-cycle is being executed. For division the setting of the NEG signals, denoted DVNEG, is defined as follows:

$$NEG0 = QS_1$$

$$NEG1 = D0DMD \cdot A_o \cdot LDQMS12 \vee D1DMD \cdot A_o \cdot LDQMS34$$

$$NEG2 = D1DMD \cdot A_o \cdot LDQMS34 \vee D2DMD \cdot A_o \cdot LDQMS56$$

$$NEG3 = D2DMD \cdot A_o \cdot LDQMS56 \vee D3DMD \cdot A_o \cdot LDQMS78$$

$$NEG4 = QS_7$$

2.11 Condition Detect

This section describes a variety of logic which detects conditions upon which control decisions are based. The condition detect logic includes register content detection such as UQ Equal Zero, and comparison and bogus result detection.

2.11.1 UQ Zero Detect

2.11.1.1 General

This logic detects the presence of all zeros in the UQ-Register and various subsections of the register. The signal UQEZ (UQ Equal Zero) is true whenever all positions of the UQ Register (UQ_i for i = 1 through 64) are zero. This signal is used to detect a zero multiplier, a zero dividend, or a zero result. The zero detection for the high order two bytes is somewhat special. The logic is subdivided into four, 4-bit groups with outputs denoted UQ14EZ (UQ bits 1 through 4 equal zero), UQ58EZ, UQ912EZ, and UQ1316EZ. These signals are used in detecting overflow and in the normalization of floating point results.

The next section describes the implementation of the logic and is followed by detailed PL/1 definition of all signals involved.

2.11.1.2 Implementation

Figure 2.11.1.2.1 illustrates the logic for the UQ-Register Zero Detect. The bulk of the logic consists of the two variants of diode matrix boards: the 234-20 and 234-21. The 234-20 consists of two, 8-input ANDS and 2 inputs for an OR. The 234-21 consists of four, 4-input ANDS and 2 inputs for an OR. The detailed logic is shown on Drawing 211-01.

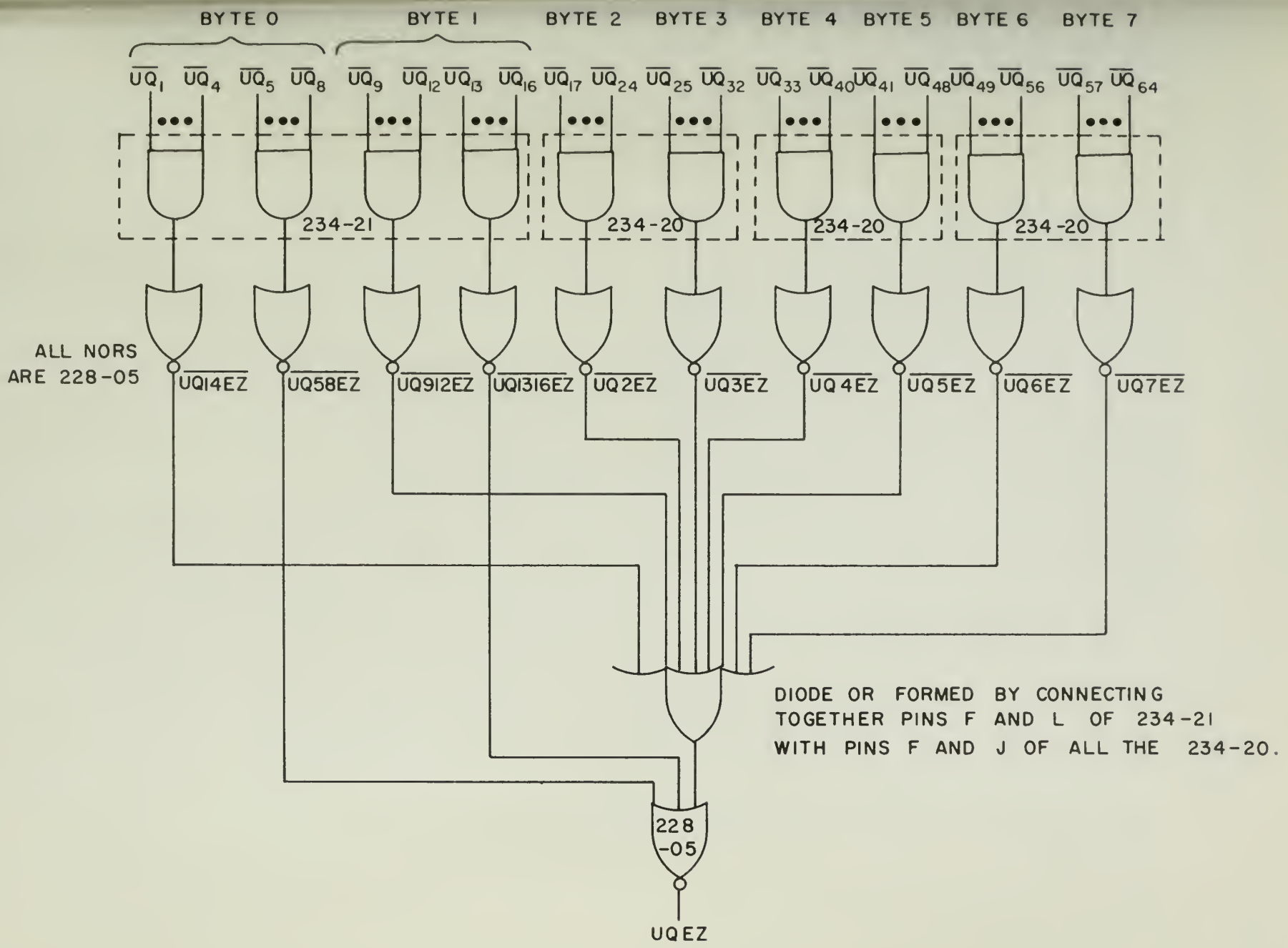


Figure 2.11.1.2.1 - Logic for UQ-Register Zero Detect

```
UQEZ:       /*CONDITION DETECT FOR THE CONTENTS OF THE UQ-REGISTER
              EQUAL ZERO */
            PROCEDURE BIT(1);
            DECLARE(UQ14EZ, UQ58EZ, UQ912EZ, UQ1316EZ, UQ2EZ,UQ3EZ,
                    UQ4EZ,UQ5EZ,UQ6EZ,UQ7EZ) BIT(1);
/*UQ14EZ */IF SUBSTR(UQ,1,4)='0000'B THEN UQ14EZ /*UQ BITS 1-4 EQUAL ZER
                                  ='1'B ELSE UQ14EZ='0'B;
/*UQ58EZ */IF SUBSTR(UQ,5,4)='0000'B THEN UQ58EZ /*UQ BITS 5-8 EQUAL ZER
                                  ='1'B ELSE UQ58EZ='0'B;
/*UQ912EZ*/IF SUBSTR(UQ,9,4)='0000'B THEN UQ912EZ /*UQ BITS 9-12 EQUAL Z
                                  ='1'B ELSE UQ912EZ='0'B;
/*UQ1316EZ*/IF SUBSTR(UQ,13,4)='0000'B THEN UQ1316EZ /*UQ BITS 13-16 EQU
                        7ERO*/ ='1'B ELSE UQ1316EZ='0'B;
/*UQ2EZ   */IF SUBSTR(UQ,17,8)=(8'0'B THEN UQ2EZ /*UQ BYTE 2 EQUAL ZERO*/
                                  ='1'B ELSE UQ2EZ ='0'B;
/*UQ3EZ   */IF SUBSTR(UQ,25,8)=(8)'0'B THEN UQ3EZ /*UQ BYTE 3 EQUAL ZERO*
                                  ='1'B ELSE UQ3EZ ='0'B;
/*UQ4EZ   */IF SUBSTR(UQ,33,8)=(8)'0'B THEN UQ4EZ /*UQ BYTE 4 EQUAL ZERO*
                                  ='1'B ELSE UQ4EZ ='0'B;
/*UQ5EZ   */IF SUBSTR(UQ,41,8)=(8)'0'B THEN UQ5EZ /*UQ BYTE 5 EQUAL ZERO*
                                  ='1'B ELSE UQ5EZ ='0'B;
/*UQ6EZ   */IF SUBSTR(UQ,49,8)=(8)'0'B THEN UQ6EZ /*UQ BYTE 6 EQUAL ZERO*
                                  ='1'B ELSE UQ6EZ='0'B;
/*UQ7EZ   */IF SUBSTR(UQ,57,8)=(8)'0'B THEN UQ7EZ /*UQ BYTE 7 EQUAL ZERO*
                                  ='1'B ELSE UQ7EZ='0'B;
            UQEZ = UQ14EZ & UQ58EZ & UQ912EZ & UQ1316EZ & UQ2EZ & UQ3EZ
                 & UQ4EZ & UQ5EZ & UQ6EZ & UQ7EZ;
            RETURN(UQEZ);
            END;
```

2.11.2 UH Zero Detect

2.11.2.1 General

This logic is identical to the UQ Zero Detect except that it is driven by the UH-Register rather than the UQ-Register. This logic is used to detect a zero multiplicand or a zero divisor. The subsignals UH14EZ, UH58EZ, UH912EZ and UH1316EZ are used in detecting overflow.

2.11.2.2 Implementation

Figure 2.11.2.2.1 illustrates the logic for the UH-Register Zero Detect. It is identical to the UQ Zero Detect Logic. The detailed logic is shown on Drawing 211-01.

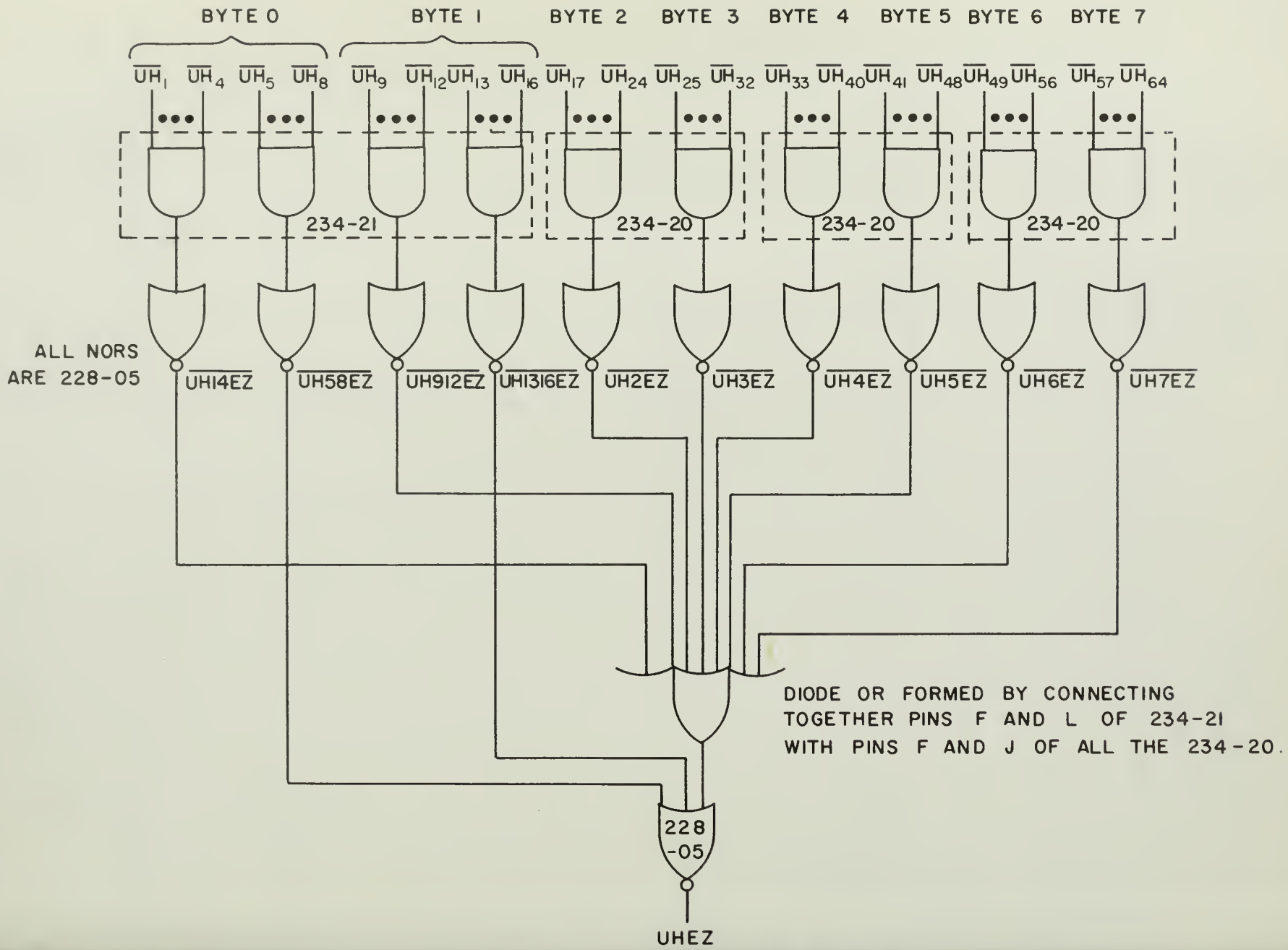


Figure 2.11.2.2-1. Logic for UH-Register Zero Detect

```
HEZ:      /*CONDITION DETECT FOR THE CONTENTS OF THE UH-REGISTER
            EQUAL ZERO */
          PROCEDURE BIT(1):
          DECLARE(UH14EZ, UH58EZ, UH912EZ, UH1316EZ, UH2EZ,UH3EZ,
                  UH4EZ,UH5EZ,UH6EZ,UH7EZ) BIT(1):
*UH14EZ */IF SUBSTR(UH,1,4)='0000'B THEN UH14EZ /*UH BITS 1-4 EQUAL ZERO*/
                              ='1'B ELSE UH14EZ='0'B;
*UH58EZ */IF SUBSTR(UH,5,4)='0000'B THEN UH58EZ /*UH BITS 5-8 EQUAL ZERO*/
                              ='1'B ELSE UH58EZ='0'B;
*UH912EZ*/IF SUBSTR(UH,9,4)='0000'B THEN UH912EZ /*UH BITS 9-12 EQUAL ZERO*/
                              ='1'B ELSE UH912EZ='0'B;
*UH1316EZ*/IF SUBSTR(UH,13,4)='0000'B THEN UH1316EZ /*UH BITS 13-16 EQUAL
                         ZERO*/ ='1'B ELSE UH1316EZ='0'B;
*UH2EZ   */IF SUBSTR(UH,17,8)=(8)'0'B THEN UH2EZ /*UH BYTE 2 EQUAL ZERO*/
                              ='1'B ELSE UH2EZ ='0'B;
*UH3EZ   */IF SUBSTR(UH,25,8)=(8)'0'B THEN UH3EZ /*UH BYTE 3 EQUAL ZERO*/
            ='1'B ELSE UH3EZ ='0'B;
*UH4EZ   */IF SUBSTR(UH,33,8)=(8)'0'B THEN UH4EZ /*UH BYTE 4 EQUAL ZERO*/
                              ='1'B ELSE UH4EZ ='0'B;
*UH5EZ   */IF SUBSTR(UH,41,8)=(8)'0'B THEN UH5EZ /*UH BYTE 5 EQUAL ZERO*/
                              ='1'B ELSE UH5EZ ='0'B;
*UH6EZ   */IF SUBSTR(UH,49,8)=(8)'0'B THEN UH6EZ /*UH BYTE 6 EQUAL ZERO*/
                              ='1'B ELSE UH6EZ='0'B;
*UH7EZ   */IF SUBSTR(UH,57,8)=(8)'0'B THEN UH7EZ /*UH BYTE 7 EQUAL ZERO*/
                              ='1'B ELSE UH7EZ='0'B;
            UHEZ = UH14EZ & UH58EZ & UH912EZ & UH1316EZ & UH2EZ& UH3EZ
                & UH4EZ & UH5EZ & UH6EZ & UH7EZ;
            RETURN(UHEZ):
            END:
```

2.11.3 UQ All Ones Detect

2.11.3.1 General

This logic detects the presence of all ones in certain subsections of the UQ Register. The signal UQ03AØ is true whenever all positions of byte 0 through byte 3 of the UQ are one. This signal is used primarily in detecting overflow of negative integers. The signal UQ45AØ is true whenever all positions of byte 4 and byte 5 are one.

2.11.3.2 Implementation

Figure 2.11.3.2.1 illustrates the logic for the all ones detect. It is essential the same as the UQ Zero Detect logic except that the inputs are true rather than complement side of the UQ Register.

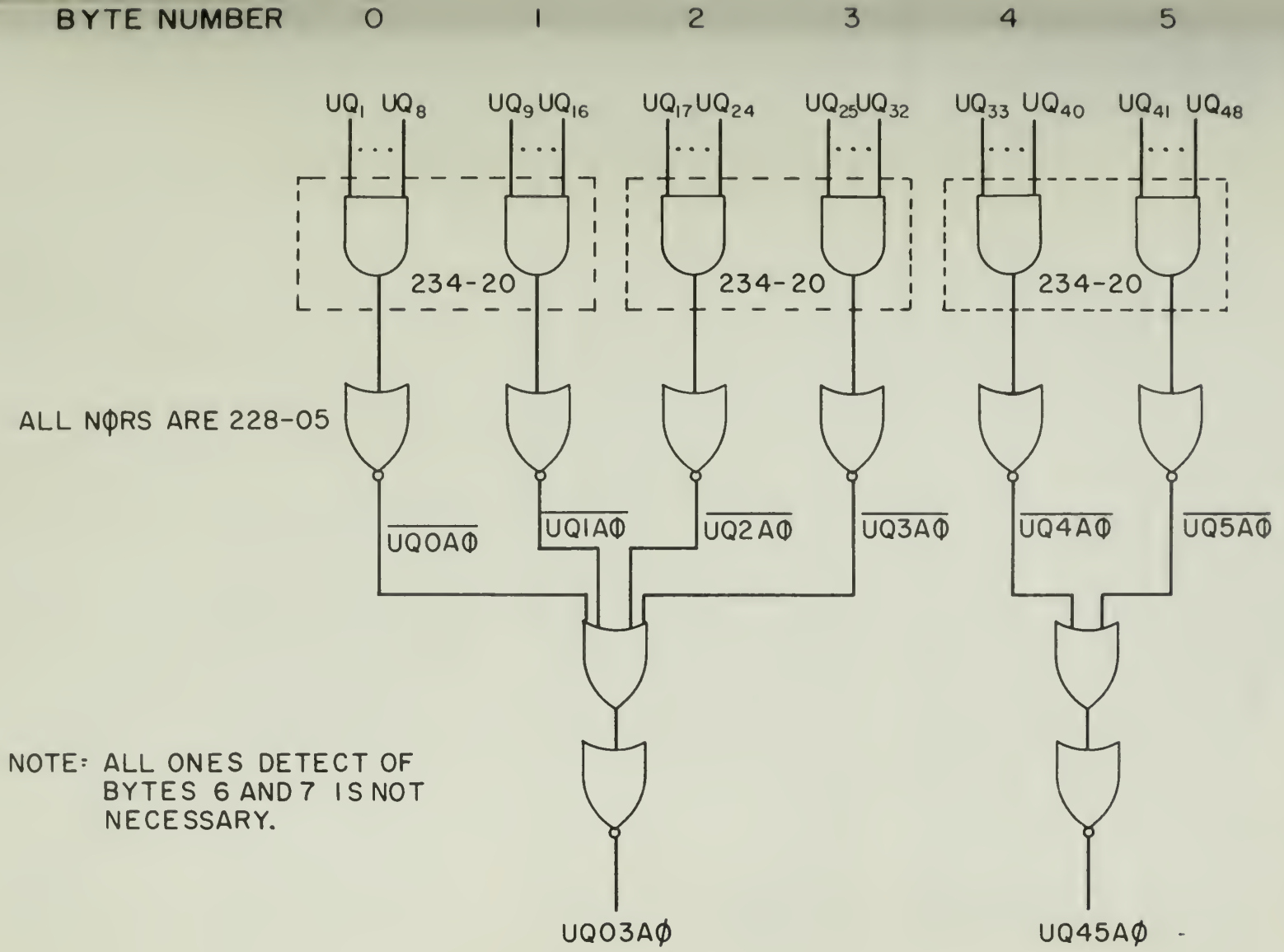


Figure 2.11.3.2.1 - Logic for UQ Register (Bytes 0-5) All Ones Detect

```
UQ03A0:     /*CONDITION DETECT FOR THE CONTENTS OF BYTES 0-3 OF THE
              UQ REGISTER ALL ONES */
            PROCEDURE BIT(1);
            DECLARE(UQ0A0,UQ1A0,UQ2A0,UQ3A0) BIT(1);
/*UQ0A0  */IF SUBSTR(UQ,1,8)=(8)'1'B THEN UQ0A0 /*UQ BYTE 0 ALL ONES*/
                                   ='1'B ELSE UQ0A0 ='0'B;
/*UQ1A0  */IF SUBSTR(UQ,9,8)=(8)'1'B THEN UQ1A0 /*UQ BYTE 1 ALL ONES*/
                                   ='1'B ELSE UQ1A0 ='0'B;
/*UQ2A0  */IF SUBSTR(UQ,17,8)=(8)'1'B THEN UQ2A0 /*UQ BYTE 2 ALL ONES*/
                                   ='1'B ELSE UQ2A0 ='0'B;
/*UQ3A0  */IF SUBSTR(UQ,25,8)=(8)'1'B THEN UQ3A0 /*UQ BYTE 3 ALL ONES*/
                                   ='1'B ELSE UQ3A0 ='0'B;
            UQ03A0 = UQ0A0 & UQ1A0 & UQ2A0 & UQ3A0;
            RETURN(UQ03A0);
            END;
UQ45A0:      /*CONDITION DETECT FOR THE CONTENTS OF BYTES 4-5 OF THE
               UQ REGISTER ALL ONES*/
            PROCEDURE BIT(1);
            DECLARE (UQ4A0,UQ5A0) BIT(1);
/*UQ4A0  */IF SUBSTR(UQ,33,8)=(8)'1'B THEN UQ4A0 /*UQ BYTE 4 ALL ONES*/
                                   ='1'B ELSE UQ4A0 ='0'B;
/*UQ5A0  */IF SUBSTR(UQ,41,8)=(8)'1'B THEN UQ5A0 /*UQ BYTE 5 ALL ONES*/
                                   ='1'B ELSE UQ5A0 ='0'B;
            UQ45A0 = UQ4A0 & UQ5A0;
            RETURN (UQ45A0);
            END;
```

2.11.4 Flag Registers

2.11.4.1 General

The data structure of Illiac III includes a flag bit with each byte. As arithmetic operands are loaded into registers of an arithmetic unit the flags of operands (A-op and B-op) are stored in the FA and FB registers, respectively. The flags of the result are normally taken from the flags of the FA register in accordance with the conventions described in Section 1.5. For a comparison or bogus condition the flags of the result are not the flags of the A-Operand but rather are set as an indication of the result of the comparison or the nature of the bogus condition. The setting of the indicators (DV, LS, GT, EQ, LT, FM, UM and ID) are loaded into the FA-Register.

The flags of each word of operand are brought in on lines V_{19}, V_{29}, V_{39} and V_{49} of the V-BUS. These bits may be loaded into either the right or left four bits of either the FA-Register or FB-Register. The input path to the FA-Register includes a two-way selector: one path allows the flags from the V-BUS to be stored; the other allows the condition indicators to be stored. The FB-Register requires no input selector since it is loaded only from the V-BUS. The output of the FA Selector is denoted FASEL. The signals associated with these operations are described below and defined with PL/1 notation in Section 2.11.4.3.

Signal Name	Description
FAILFA	Select the flags of the A1 operand (the first word of the A operand) to the left half of the FA-Register. Once selected, the flags are loaded into the flip-flops under control of LDFAL (Load FA Left).

Signal Name	Description
FA2RFA	Select the flags of the A2 operand (the second word of the A operand) to the right half of the FA Register. Once selected, the flags are loaded into the flip-flops under control of LDFAR (Load FA Right).
FB1LFB	Select and load the flags of the B1 operand (the first word of the B Operand) into the left half of the FB-Register.
FB2RFB	Select and load the flags of the B2 operand (the second word of the B Operand) into the right half of the FB-Register.
INDFA	Indicators direct to FA-Register.
LDFAL	Load FA-Register, left half, from $FASEL_1$-$FASEL_4$.
LDFAR	Load FA-Register, right half, from $FASEL_5$-$FASEL_8$.

2.11.4.2 Implementation

Figures 2.11.4.2.1 and 2.11.4.2.2 are block diagrams of the FA and FB registers, respectively. Figure 2.11.4.2.3 illustrates the logic of a typical position of the FA-Register. The FB-Register consists solely of a 260-00. Detailed logic is shown on Drawing No. 211-02.

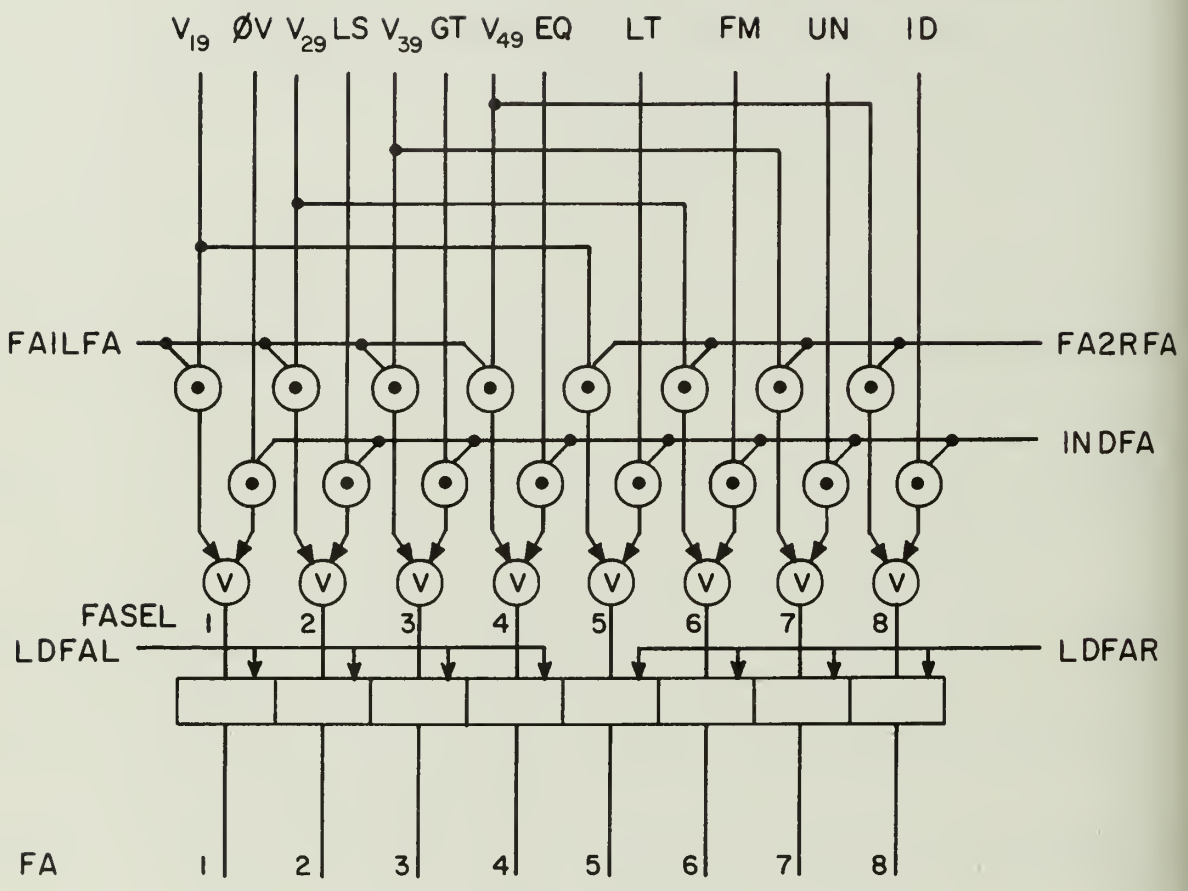


Figure 2.11.4.2.1 - Block Diagram of FA-Register

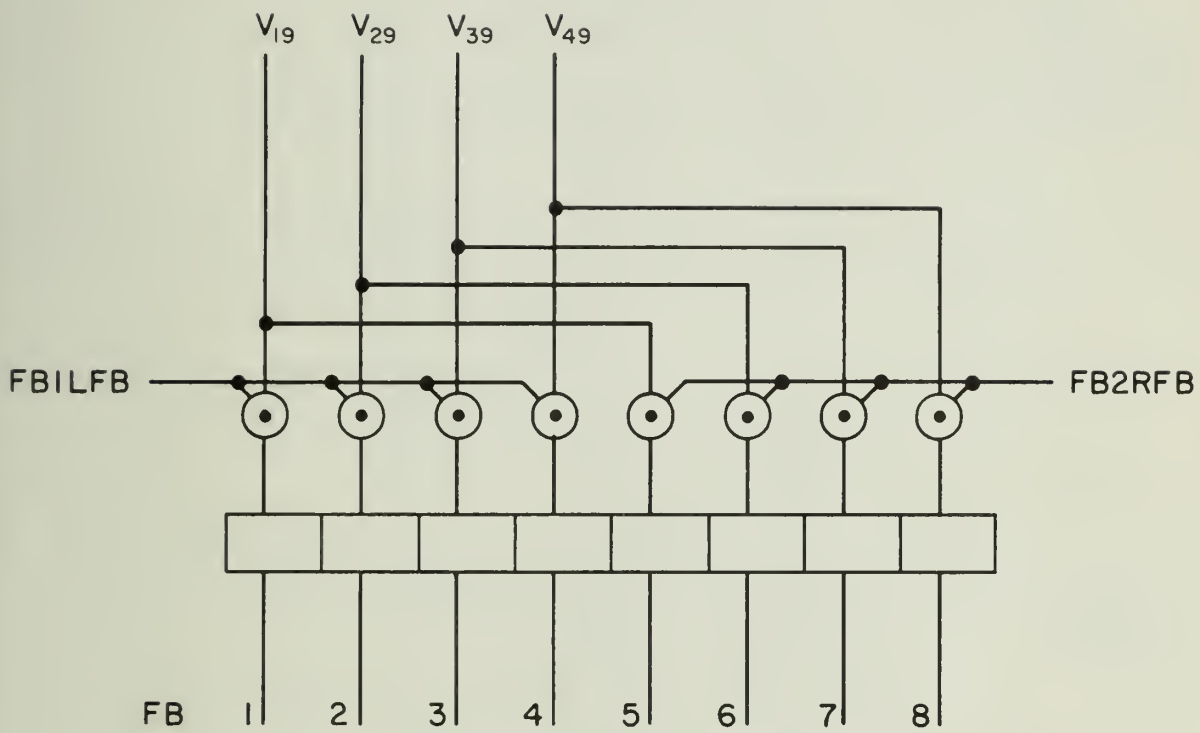


Figure 2.11.4.2.2 - Block Diagram of FB-Register

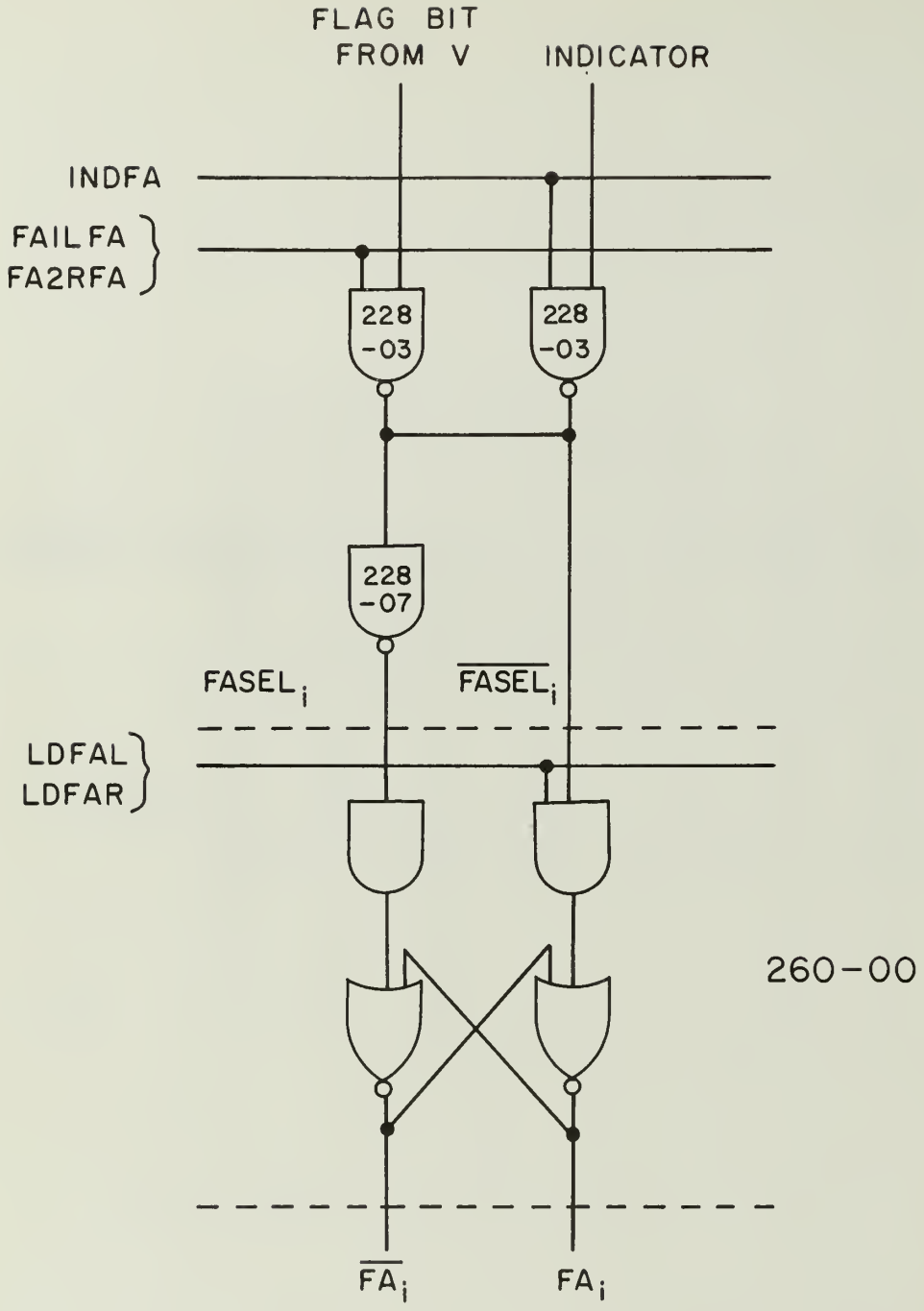


Figure 2.11.4.2.3 - Typical Position of FA-Register

2.11.4.3 PL/1 Description of Signal Names

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO FLAG REGISTERS*/
/*RELEVANT DRAWING NUMBER: 211-02 */
        DECLARE FA BIT(8); /*FLAGS OF A-OPERAND REGISTER */
        DECLARE FB BIT(8); /*FLAGS OF B-OPERAND REGISTER */
        DECLARE FASEL BIT(8);/*TRUE OUTPUT OF FA-REGISTER
                               SELECTOR*/
        DECLARE FM ENTRY RETURN(BIT(1))
A1LFA:  /*FLAGS OF WORD 1 OF A-OP TO LEFT HALF OF FA REG */
        PROCEDURE:
        SUBSTR(FASEL,1,4)=SUBSTR(V,19,1)||SUBSTR(V,29,1)||
                          SUBSTR(V,39,1)||SUBSTR(V,49,1):
        END:
A2RFA:  /*FLAGS OF WORD 2 OF A-OP TO RIGHT HALF OF FA */
        PROCEDURE;
        SUBSTR(FASEL,5,4)=SUBSTR(V,19,1)||SUBSTR(V,29,1)||
                          SUBSTR(V,39,1)||SUBSTR(V,49,1):
        END:
B1LFB:  /*FLAGS OF WORD 1 OF B-OP TO LEFT HALF OF FB*/
        PROCEDURE:
        SUBSTR(FB,1,4)=SUBSTR(V,19,1)||SUBSTR(V,29,1)||
                       SUBSTR(V,39,1)||SUBSTR(V,49,1);
        END:
B2RFB:  /*FLAGS OF WORD 2 OF B-OP TO RIGHT HALF OF FB*/
        PROCEDURE:
        SUBSTR(FB,5,4)=SUBSTR(V,19,1)||SUBSTR(V,29,1)||
                       SUBSTR(V,39,1)||SUBSTR(V,49,1);
        END:
M:      PROCEDURE BIT(1); /* FLAG MATCH DETECTION */
        DECLARE TEMP BIT(1);
        IF((FA&FB)|(¬FA&¬FB))='11111111'B THEN TEMP='1'B;
        ELSE TEMP='0'B;
        RETURN (TEMP);
        END;
NDFA:   /*SELECT INDICATORS DIRECT TO FA */
        PROCEDURE;
        FASEL= GT||EQ||LT||OV||FM||UN||LS||ID;
        END;
LFAL:   /*LOAD LEFT HALF OF FA */
        PROCEDURE;
        SUBSTR(FA,1,4)=SUBSTR(FASEL,1,4):
        END:
LFAR:   /*LOAD RIGHT HALF OF FA */
        PROCEDURE;
        SUBSTR(FA,5,4)=SUBSTR(FASEL,5,4);
        END;
```

2.11.5 Flag Match

2.11.5.1 General

A flag bit is associated with each byte of the operands. As the operands are loaded into AU registers the flags of the A and B operands are stored in the FA and FB registers, respectively. In the course of a compare operation (CPRA) the flags of the two operands are also tested for identity. If the flags of the two operands are identical the Flag Match (FM) signal is set to one. If Π denotes a Boolean product, m equals the number of bytes per operand, and '$\equiv$' denotes the identity operator thus

$$FM = \prod_{i=1}^{m} (FA_i \equiv FB_i) .$$

2.11.5.2 Implementation

Figure 2.11.5.2.1 illustrates the Flag Match logic. Detailed logic is shown on Drawing No. 211-02.

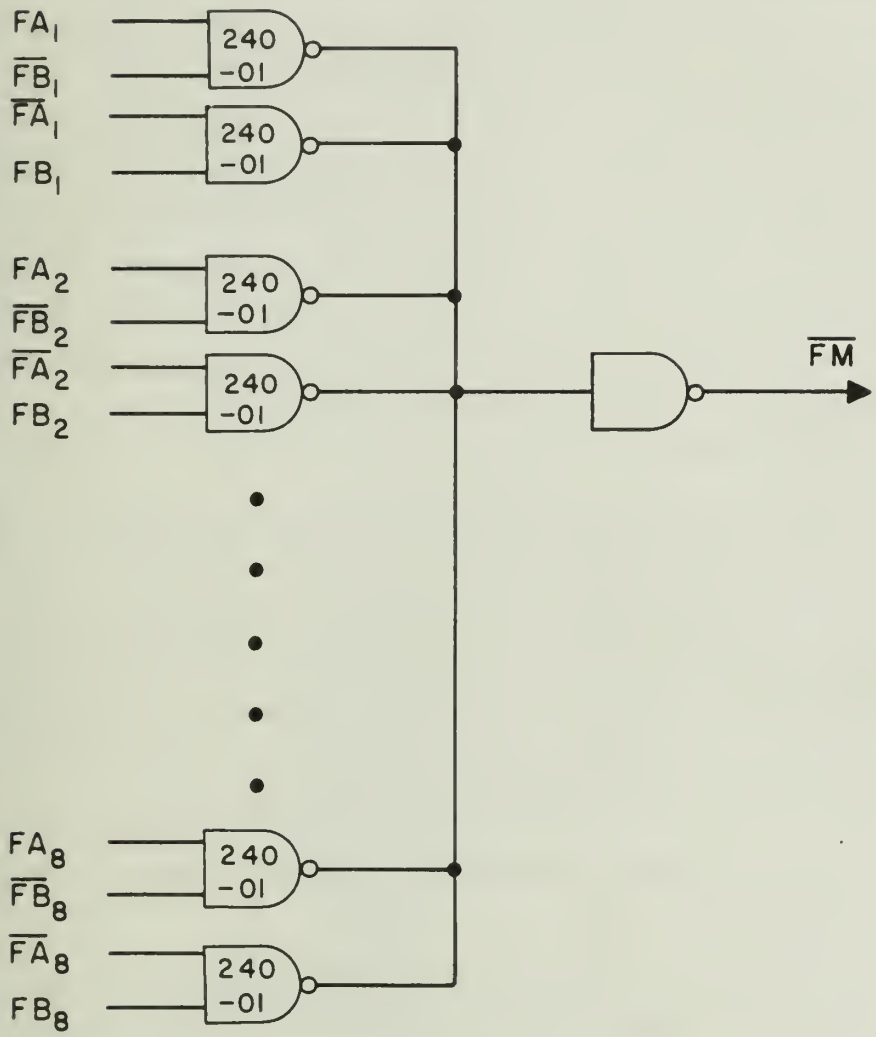


Figure 2.11.5.2.1 - Flag Match Logic

2.12 Exponent Arithmetic Unit

2.12.1 General

The Exponent Arithmetic Unit (EAU) performs arithmetic on the exponents of floating point operands during floating point (Number Type = 10) operations. For addition, subtraction and comparison, the difference of the exponent is formed and used to control a shift loop which aligns the radix points of the fractional part of the two operands. The exponent of the result is the same as the larger exponent of the two operands. For multiplication and division no alignment is required. The exponent of a product is the sum of the exponents of the operands. The exponent of a quotient is the difference of the exponent of the dividend and divisor. The role of the EAU in each floating point operation is defined in detail in Section 3.4.

The exponents are expressed in an excess 64 notation. The bit representation for an exponent value, X (base 10) in excess 64 notation may be formed by adding 64 (base 10) to X (base 10) and converting the result to the binary equivalent. A consequence of this notation is that the high order bit is 1 for positive or zero values and 0 for negative values. The range of allowable exponent values -64 through +63. Table 2.12.1.1 defines the representation for all exponent values.

The EAU is implemented with the Texas Instruments 7400 Series Dual Inline packaged logic.

Figure 2.12.1.1 is a block diagram of the EAU. The bulk of Section 2.12 will be a description of each of the blocks of this figure.

Table 2.12.1.1 - Excess-64 Notation Used in Exponent Unit

Magnitude	Positive Representation	Negative Representation
0	1000000	None
1	1000001	0111111
2	1000010	0111110
3	1000011	0111101
4	1000100	0111100
5	1000101	0111011
6	1000110	0111010
7	1000111	0111001
8	1001000	0111000
9	1001001	0110111
10	1001010	0110110
11	1001011	0110101
12	1001100	0110100
13	1001101	0110011
14	1001110	0110010
15	1001111	0110001
16	1010000	0110000
17	1010001	0101111
18	1010010	0101110
19	1010011	0101101
20	1010100	0101100
21	1010101	0101011
22	1010110	0101010
23	1010111	0101001
24	1011000	0101000
25	1011001	0100111
26	1011010	0100110
27	1011011	0100101
28	1011100	0100100
29	1011101	0100011
30	1011110	0100010
31	1011111	0100001
32	1100000	0100000

Table 2.12.1.1 - (Continued) - Excess-64 Notation Use in Exponent Unit

Magnitude	Positive Representation	Negative Representation
33	1100001	0011111
34	1100010	0011110
35	1100011	0011101
36	1100100	0011100
37	1100101	0011011
38	1100110	0011010
39	1100111	0011001
40	1101000	0011000
41	1101001	0010111
42	1101010	0010110
43	1101011	0010101
44	1101100	0010100
45	1101101	0010011
46	1101110	0010010
47	1101111	0010001
48	1110000	0010000
49	1110001	0001111
50	1110010	0001110
51	1110011	0001101
52	1110100	0001100
53	1110101	0001011
54	1110110	0001010
55	1110111	0001001
56	1111000	0001000
57	1111001	0000111
58	1111010	0000110
59	1111011	0000101
60	1111100	0000100
61	1111101	0000011
62	1111110	0000010
63	1111111	0000001
64	None	0000000

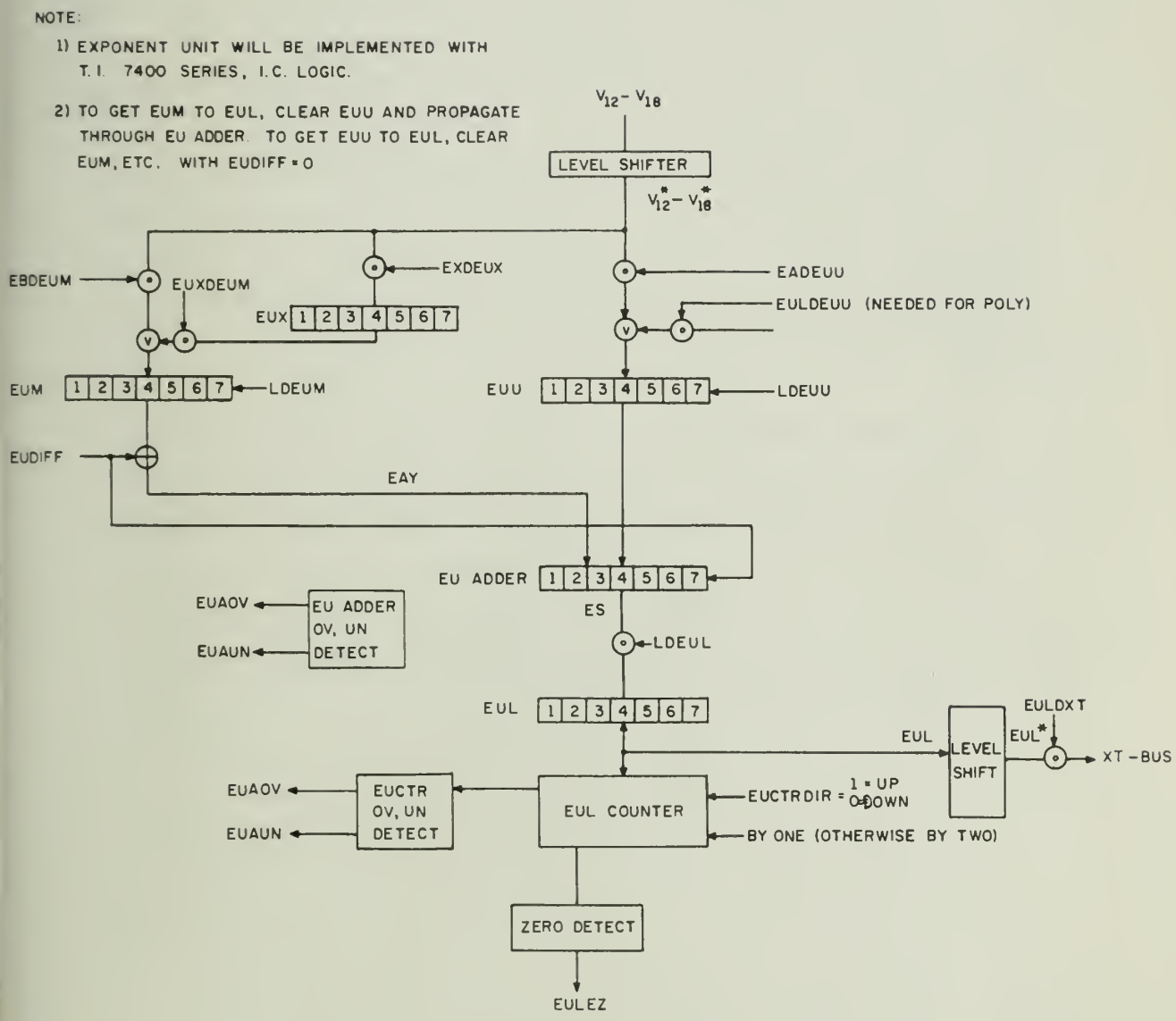


Figure 2.12.1.1 - Block Diagram of Exponent Arithmetic Unit

2.12.2 Level Shifters

Since the 1018-DTL logic and the Texas Instruments TTL logic have somewhat different operating ratings, attention must be given to connections between the two. The supply voltage to the TTL logic, V_{CC}, is restricted to a range between 4.9 and 5.6 volts.* The logical one input to the TTL logic should not exceed V_{CC} and therefore the nominal +6 logical one output of the DTL logic must be reduced before it drives TTL logic. All TTL logic to be driven by DTL logic will be mounted on the universal 1018-299 board. The conversion from DTL to TTL logic is accomplished by placing a diode on the 299 board for each pin which is to be used as an input from DTL logic. This forward biased provides adequate level shifting. The Level Shifter box in the upper right portion of Figure 2.12.1.1 consists merely of these diodes. The signals $V_{12}^* - V_{13}^*$ at the output of the shifter correspond to $V_{12} - V_{18}$ at the input.

The conversion from TTL to DTL logic requires no intervening components but is subject to constraints. The TTL logic must be capable of sinking all the current required to hold the input of the DTL logic at "0". Most TTL units will sink 16 ma. and thus could drive up to four DTL units. The Level Shifter in the lower left hand portion of Figure 2.12.1.1 therefore represents no hardware: it merely denotes the interface between the two types of logic.

---

*R. Borovec - Internal Memo to Illiac III Engineering Staff, August 1, 1969.

2.12.3 Registers

The Exponent Arithmetic Unit includes four registers for storing seven bit exponents and results. The accumulator is the EUU-Register (Exponent Unit Upper Register) which is analogous to the US-UM Registers of the main arithmetic unit. Note, however, that a redundant representation is not used in the exponent unit as it is in the main unit and therefore only one storage position per digit is required. The secondary rank of the accumulator is the EUL-Register (Exponent Unit Lower Register). It is analogous to the LS-LM Registers of the main AU. The EUL-Register is also an integral part of an up-down counter which is used to control the right shifting required for pre-alignment of the radix point. The EUU Register supplies one operand to the adder; the EUM Register supplies the other. The EUM Register (Exponent Unit M-Register) is analogous to the M-Register of the main AU. The EU also includes an auxiliary register, the EUX-Register. It is used in the polynomial evaluation routine which evaluates polynomials, P(X). This register stores the exponent of X and thereby frees the EUM and EUU Registers for storing the exponent of the partial result and the current exponent of the current coefficient.

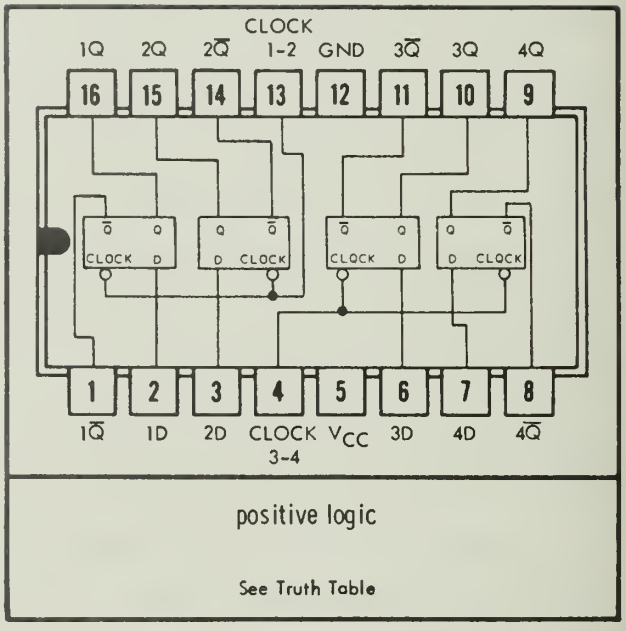
The storage for all but the EUL-Register is implemented with the Texas Instruments SN7475N Quadruple Bistable Latch shown in Figure 2.12.3.1. (from Texas Instruments New Products Bulletin SC9605, February 1967). Information present at a data (D) input is transferred to the Q output when the clock is high, and the Q output will follow the data input as long as the clock remains high. When the clock goes low, the information (that was present at the data input at the time the transition occurred is retained at the Q output (latched) until the clock is permitted to go high. The clock pulse width should be at least 30 ns. The clock signal will be referred to as a "Load" signal in discussions of the application in the arithmetic units.

The EUL-Register is implemented with the SN7474 Dual D-Type Edge-Triggered flip-flop. This flip-flop will be described in discussion of the EUL-Register and EUL Counter.

logic

TRUTH TABLE (Eoch Latch)		
t_n	t_{n+1}	
D	Q	$\overline{Q}$
1	1	0
0	0	1

NOTES: 1. t_n = bit time before clock pulse.
2. t_{n+1} = bit time after clock pulse.



description

The SN7475N is a monolithic, quadruple, bistable lotch with complementary Q and $\overline{Q}$ outputs. Information present at a dato (D) input is transferred to the Q output when the clock is high, and the Q output will follow the data input as long as the clock remains high. When the clock goes low, the information (that was present at the data input at the time the transition occurred) is retained at the Q output until the clock is permitted to go high. Pin assignments, or physical placement of the logical functions, were selected to coincide with the physical placement of logicol functions of other Series 74 circuits which are most likely to be used as inputs to, or outputs from, the SN7475N.

This latch is ideally suited for use as temporory storoge for binary information between processing units and input/output or indicator units. Applications are shown for the SN7475N being used for temporary storage of 4-bit binary data and as a dual master-slave flip-flop with two-phase clocking.

(From Texas Instrument 1967-68 Integrated Circuits Catalog)

Figure 2.12.3.1 T. I. SN7475N Quadruple Bistable Latch

2.12.3.1 EUM-Register

The EUM (Exponent Unit M-Register) Register consists of 7 bits of storage and seven positions of a two-way selector. It also includes complementing logic on its output. This register stores the exponent of the B-Operand. The contents of EUU may be combined by addition or subtraction with the contents of the EUU Register.

The storage for the EUM-Register is implemented with two packages of the SN747SN Quadruple Bistable Latch previously described in Section 2.12.3. A two-way selector into the latch provides either V_{12}* through V_{18}* by selecting EBDEUM, the output of the EUX-Register by selecting EUXDEUM, or all zeros (for reset) by selecting neither. The output of EUM drives a set of exclusive -OR (complementing logic) gates which in turn drive an input to the adder (EAY). When EUDIFF is zero, the true contents of the EUM drive the adder; when EUDIFF is one, the complement of the contents of the EUM drives the adder. This facility is used in performing subtraction of exponents.

Figure 2.12.3.1.1 illustrates a typical position of the EUM-Register and Selectors. Note that the output of the selector (EUMSEL) which drives the "D" input to the quad latch is in the complement form. The logical interpretation of the outputs of the latch are therefore interchanged. The $\overline{Q}$ output corresponds to EUM while the Q output corresponds to Q. Detail logic is shown on Drawing No. 212-02.

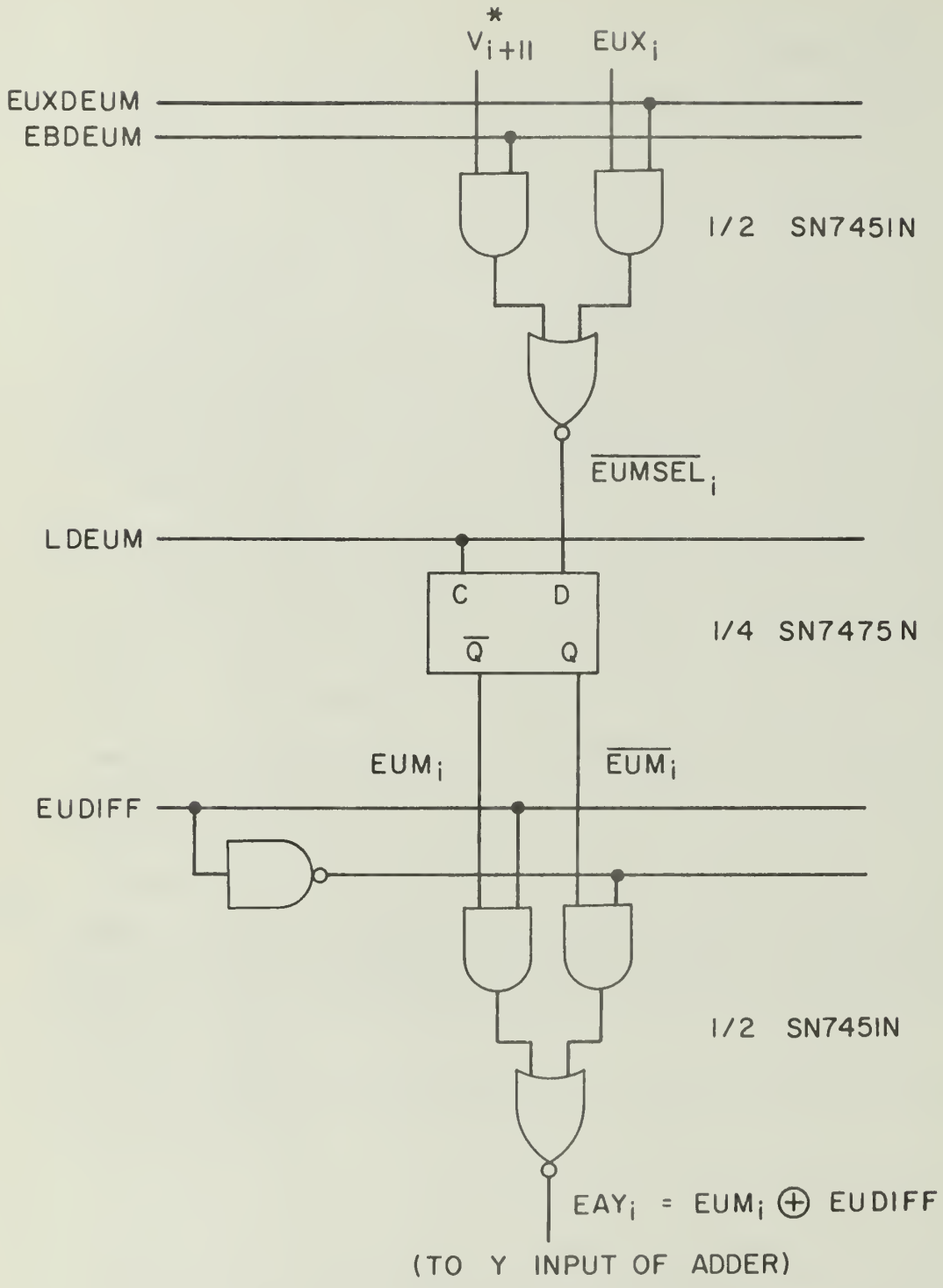


Figure 2.12.3.1.1 - Typical Position of EUM Register and Selector

2.12.3.2 EUU-Register

The EUU-Register (Exponent Unit Upper Register) consists of seven bits of storage and seven positions of a two-way selector. It is the primary rank of the accumulator. It initially stores the exponent of the A-Operand by the selection of EADEUU. The signal EULDEUU selects the contents of the secondary rank of the accumulator (EUL) for return to the EUU register. This path is necessary in performing the polynomial evaluation; the exponent of the partial result is gated back to the EUU for combination with the exponent of the next coefficient.

The EUU-Register/Selector is logically identical to the EUM Register/Selector except that no complementing logic on the outputs is required. Figure 2.12.3.2.1 is a typical position of the EUU Register and Selectors. Detailed logic is shown on Drawing No. 212-02.

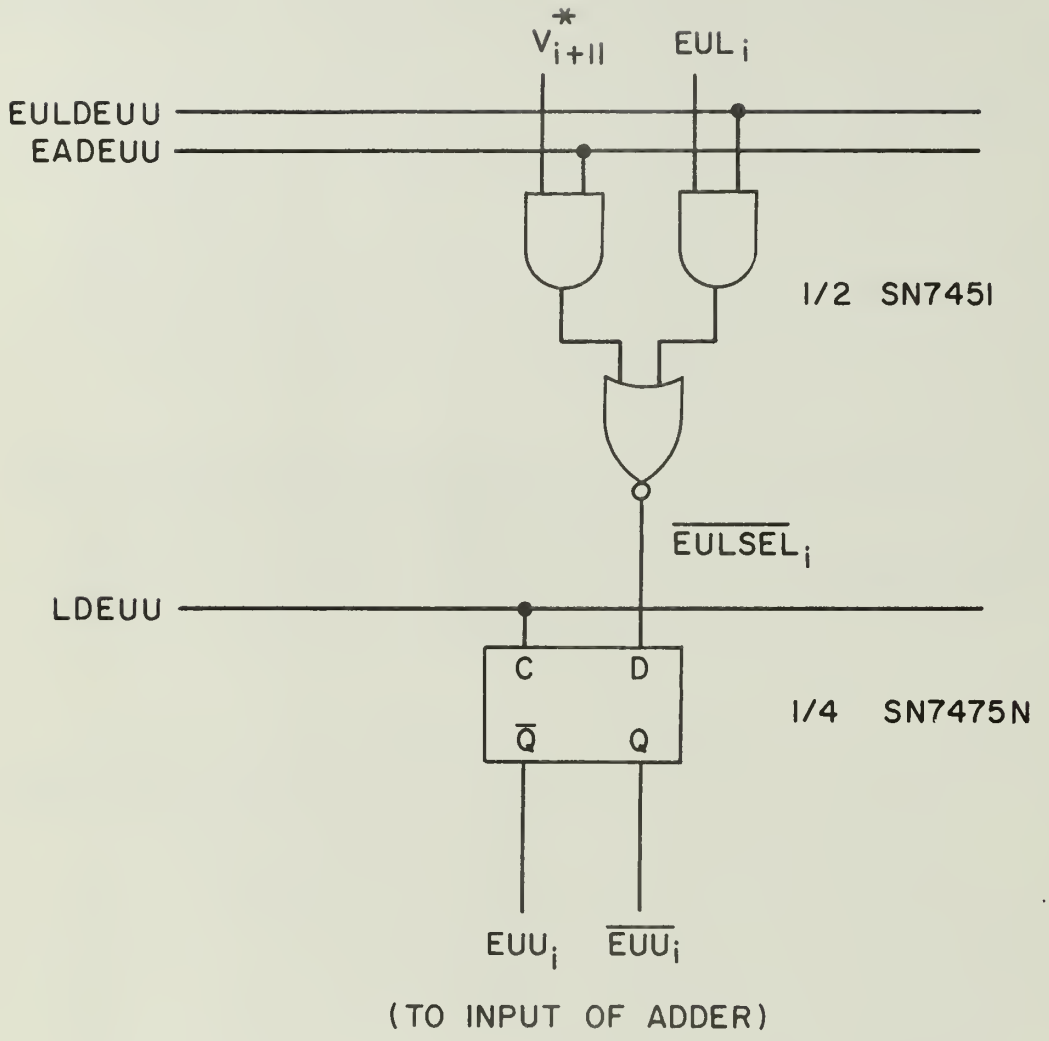


Figure 2.12.3.2.1 - Typical Position of EUU Register and Selector

2.12.3.3 EUX Register

The EUX Register is an auxiliary register required for the storing of the exponent of the argument, X, of the polynomial, P(X), during POLY. It is loaded from V_{12}^* - V_{18}^* and drives the EUM Selector.

The EUX-Register is implemented with the SN7475N Quadruple Latch. Figure 2.12.3.3.1 illustrates a typical position. Detailed logic is shown on Drawing No. 212-02.

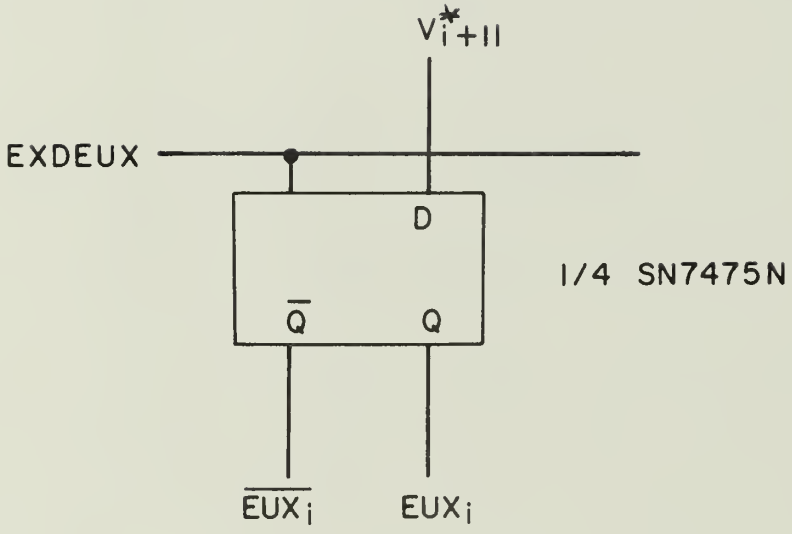


Figure 2.12.3.3.1 - Typical Position of EUX-Register

2.12.3.4 EUL-Register

The EUL Register is an integral part of the EUL Counter. It is described in Section 2.12.6.

2.12.4 Exponent Unit Adder

The Exponent Unit Adder is implemented with one package of the Texas Instruments - SN7483N four bit adder, and two packages of the Texas Instruments - SN7482N two bit adders. These packages are carry-ripple full adders. The logic equations for the ith position of the adder are as follows:

Let A_i and B_i be the ith bits of the two numbers to be added; Σ_i the sum bit; C_{in}, the carry into the position; and C_{out}, the carry out. Then:

$$\Sigma_i = (A_i \oplus B_i) \quad \overline{C}_{in} \vee (A_i \equiv B_i) \quad C_{in}$$

$$C_{out} = A_i B_i \vee C_{in} (A_i \oplus B_i)$$

Note that C_{out} from stage i is C_{in} to stage i-1.

The high-order position, position 1, of the adder requires some special attention since the operands are represented in an excess 64 notation. Recall that a number with decimal value, x, is converted to excess 64 notation, denoted x', by adding 64, thus x' = x + 64. Similarly for another number, y, y' = y + 64. The sum of x' and y' should also be in excess 64 notation, thus x' + y' should be x + y + 64. Similarly for tion of x' and y', x' + y' = x + 64 + y + 64 = x + y + 128. A correction is required to change the result to x + y + 64. Since the adder is only 7 bits wide, addition/subtraction is performed mod 128. The 128 in x + y + 128 is lost. A correction therefore consists of adding 64, i.e. adding a '1' in position 1.

Consider a truth table for position 1.

C_{in}	A_1	B_1	Constant of 1	Σ_1	C_{out}
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	0	0

From the above table

$$\Sigma_1 = C_{in} \cdot (A_1 \oplus B_1) \vee \overline{C}_{in} \cdot (A_1 \equiv B_1).$$

Note that this equation is identical to the general equation given above for Σ_i except that the role of C_{in} and $\overline{C}_{in}$ are interchanged. The adder will therefore perform correct excess 64 arithmetic if C_{out} of position 2 is inverted prior to becoming C_{in} to position 1.

Figure 2.12.4.1 illustrates the logic for the adder. The inputs are the outputs of the EUM complementing logic, EAY, and the output of the accumulator, EUU. The output of the adder drives the input to the EUL-Register.

Note that the carry input to the low order position, position 7, is driven by the signal EUDIFF. (Exponent Unit Difference). When the contents of EUM are to be subtracted from the contents of EUU, EUDIFF is set to 1. This causes EAY to become the ones complement of EUM and the carry into position 7 to become one. The two's complement of the contents of EUM is thereby added to the contents of EUU.

The adder logic is shown on Drawing No. 212-04.

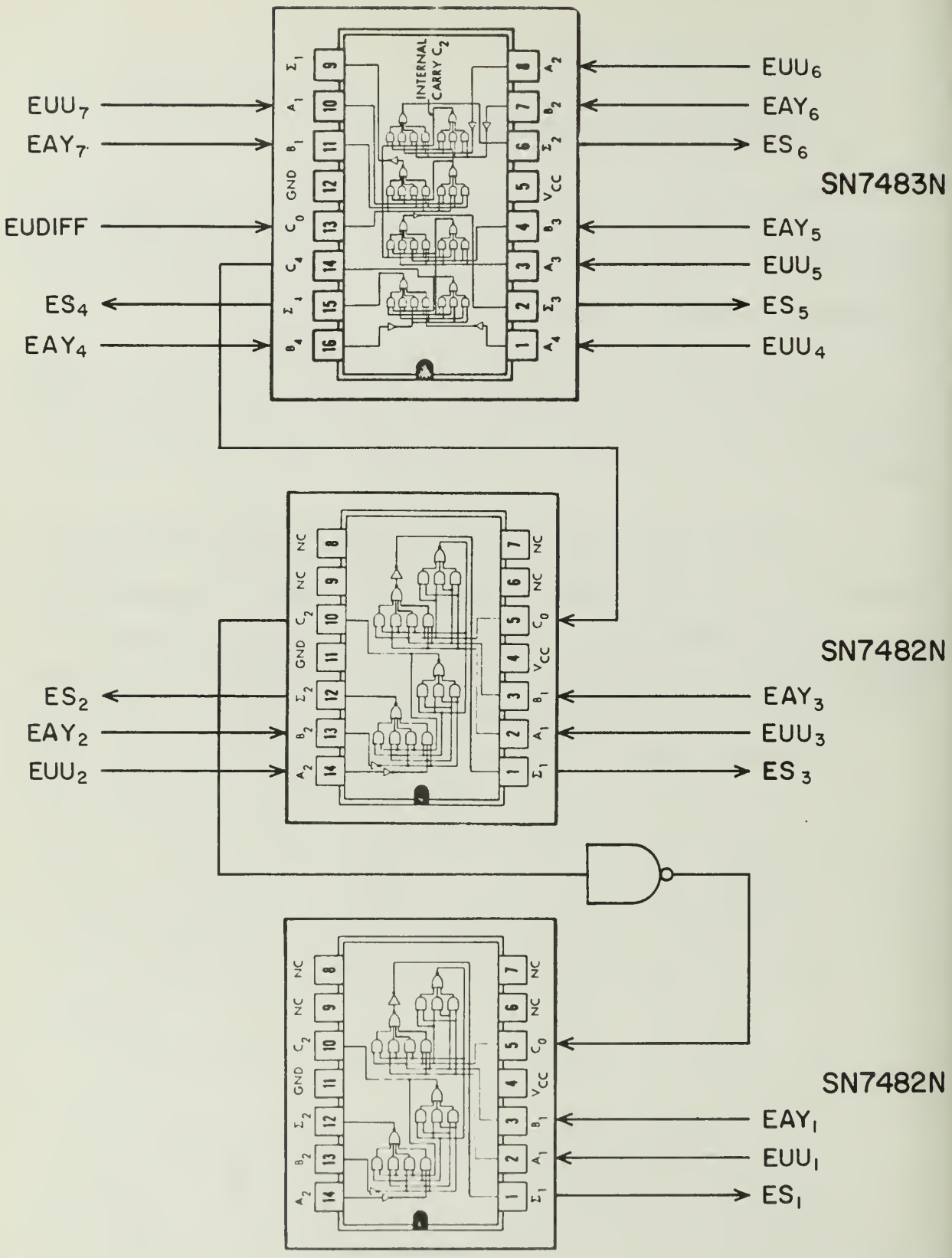


Figure 2.12.4.1 Exponent Arithmetic Unit Adder

2.12.5 Adder Overflow and Underflow Detect

The exponent values of Illiac III are in the so-called "excess 64" form as shown in Table 2.12.1.1. A seven bit pattern, A_1, A_2, A_3, A_4, A_5, A_6, A_7 has the algebraic value:

$$V(A) = (A_1-1)\ 64 + \sum_{i=2}^{7} A_i\ 2^{(7-i)}$$

Overflow Conditions

The maximum V(A) for this notation is +63. Note

$$V(A) + V(B) = \underbrace{(A_1-1+B_1-1) \cdot 64}_{(a)} \quad \underbrace{+ \sum_{i=2}^{7} (A_i + B_i) \cdot 2^{(7-i)}}_{(b)}$$

Ter (a) is max when $A_1 = B_1 = 1$. Thus overflow will occur when $A_1 = B_1 = 1$ and the value of (b) is greater than 63, i.e. when there is a carry from position 1. Thus $\emptyset V = A_1 \cdot B_1 \cdot C_{IN_1}$

Underflow Conditions

Similarly the minimum V(A) for this notation is -64. When either A_1 or B_1 (but not both) are zero then term (a) is -64 which is within this limit. Term (b) can only contribute in the positive ("more in range") direction. But when $A_1 = B_1 = 0$, term a = -128 and thus term b must be at least +64 if result is in range, i.e. there must be a carry from position 1. Thus

$$UN = \overline{A}_1 \cdot \overline{B}_1 \cdot \overline{C}_{IN_1}$$

The excess 64 arithmetic is performed with full adders with an inverter in the carry path between position 1 and 0. This has the effect of adding the two integers as if they were positive integers and then adding 64. Subtraction is performed by forming a 1's complement of the EUM input and injecting a carry into position 7.

For position one (1) of EUU adder let $C_{1in} = \overline{C_2 \text{ out.}}$

C_{2out}	C_{1in}	A_i	B_i	Σ_1	C_{1out}
1	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	0	1

$UN = \overline{C_{2out}}\ \overline{A}_1\ \overline{B}_1$

$OV = C_{2out}\ A_1\ B_1$

In terms of actual EU signal names, the overflow condition becomes

EUAUN (Exponent Unit Adder Underflow)

$$= \overline{EAY_1} \cdot \overline{EUU_1} \cdot \overline{C_2 out}$$

where $C_2 out$ is the carry out from position 2.

EUAOV (Exponent Unit Adder Overflow)

$$= EAY_1 \cdot EUU_1 \cdot C_{2out}$$

The logic for EUAUN and EUAOV is shown on Drawing No. 212-03. They each cause flip-flops to be set when they become true. Thus once an exponent overflow or underflow is detected it will remain set until a CLEAR signal is initiated.

2.12.6 EUL Counter

The EUL Counter includes a register for storing the output of the adder plus auxiliary logic to permit the contents of the EUL register to be incremented or decremented by either units of 1 or 2. In the execution of floating point addition or subtraction the exponent of the B-Operand is subtracted from the exponent of the A-Operand. If this difference, DEXP, is zero, no shifting is required. If DEXP > 0, then the fractional part of the B-Operand in the UH-Register is right shifted under control of the EUL Counter. The counter is decremented (count down) by 2 until DEXP is either 1 or 0. Each time the counter is decremented by 2 the contents of the UH-Register are shifted right 8 positions. When DEXP = 1, the counter is decremented by 1 and the contents of UH are shifted right by 4 positions. When DEXP = 0, the shifting ceases.

Similarly, if DEXP < 0, then the fractional part of the A-Operand in the UQ-Register is shifted right under control of the EUL Counter. The counter is incremented (count up) by 2 until DEXP is either -1 or 0. Each time the counter is incremented by 2 the contents of the UQ-Register are shifted right 8 positions. When DEXP = -1, the counter is incremented by 1 and the contents of UH are shifted right by 4 positions. When DEXP = 0, the shifting ceases.

Note that at most only one right shift of 4 positions is required per prealignment operation. If DEXP is an odd number, the shift of 4 is required; if DEXP is an even number, the shift is not required. The EUL counter is therefore somewhat more complex than it need be for this particular application. The counter does not need to count explicitly by 1. It was decided, however, to implement a complete counter since the cost difference was small and since the counter might be useful in future modifications.

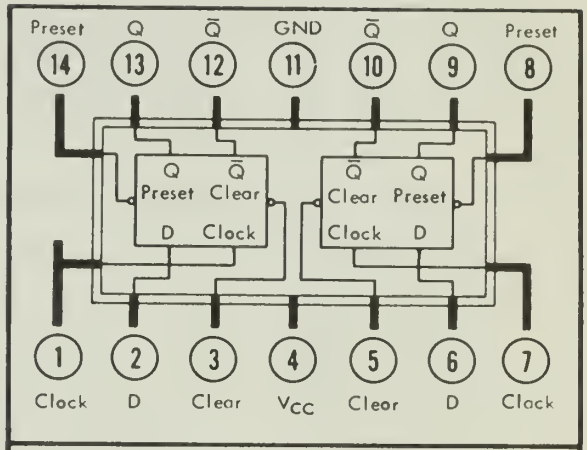
Storage for the register-counter is provided by the T.I. SN7474, Dual D-Type Edge Triggered Flip-Flop described in Figure 2.12.6.1. A typical position of the register-counter is shown in Figure 2.12.6.2. The register is loaded from the output of the

logic

TRUTH TABLE (Each Flip-Flop)

t_n	t_{n+1}	
INPUT D	OUTPUT Q	OUTPUT $\bar{Q}$
0	0	1
1	1	0

NOTES 1. t_n = bit time before clock pulse.
2. t_{n+1} = bit time after clock pulse.



positive logic

Low input to preset sets Q to logical 1
Low input to clear sets Q to logicol 0
Preset and clear are independent of clock

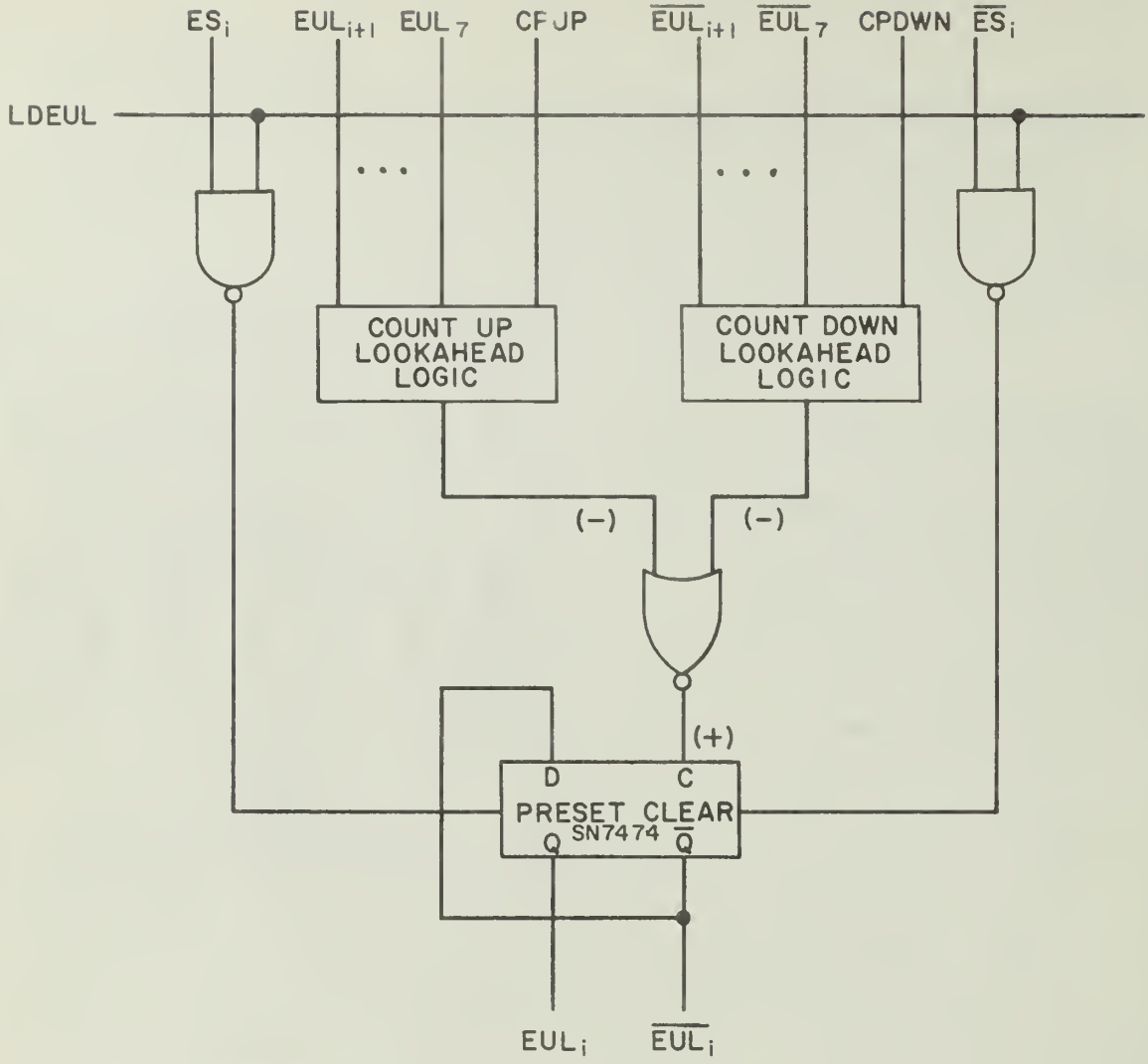
description

The SN7474 is a monolithic, dual, D-type, edge-triggered flip-flop featuring direct clear and preset inputs and complementary Q and $\bar{Q}$ outputs. Input information is transferred to the Q output on the positive edge of the clock pulse.

Clock triggering occurs at a voltage level of the clock pulse and is not directly related to the transition time of the positive going pulse. After the clock input threshold voltage has been passed the data input (D) is locked out.

The SN7474 dual flip-flop has the same clocking characteristics as the SN7470 gated (edge-triggered) flip-flop and both are ideally suited for medium- and high-speed applications. The SN7474 can be used at a significant saving in system power dissipation and package count in applications where input gating is not required.

Figure 2.12.6.1 - T.I. Type SN7474 Dual D-Type Edge-Triggered Flip-Flop



NOTE: (+) DENOTES A QUIESCENT LEVEL OF 1.
(−) DENOTES A QUIESCENT LEVEL OF 0.

Figure 2.12.6.2 - Typical Position of the EUL Register-Counter

adder, ES, by activating LDEUL. Setting of the SN7474 flip-flop by way of the Preset and Clear inputs is independent of the clock pulse. Note that the time output, Q, is coupled back to the data (D) input. The flip-flop therefore changes state each time the clock (C) signal makes a positive going transition (0 to 1). On the transition, the D input is locked out when the signal reaches a preset threshold. This property prevents a race condition and permits the flip-flop to be used as a double rank (master-slave) device.

The arrival of a clock pulse to a flip-flop is dependent upon the CPUP signal (clock pulse, up count), the CPDWN signal (clock pulse, down count), and the present count which dictates the output of the count up or count down lookahead logic. The signals CPUP and CPDWN are defined as follows:

$$\text{CPUP} = \text{CP} \cdot \text{EUCTRDIR}$$
$$\text{CPDWN} = \text{CP} \cdot \overline{\text{EUCTRDIR}}$$

where

EUCTRDIR = 1 for count up,
EUCTRDIR = 0 for count down.

Quiescently, CP = 0, thus CPUP = CPDWN = 0 and therefore the C (clock) inputs to all flip-flops of the counter are at logical 1 (See Figure 2.12.6.2). Recall that the flip-flop is triggered by the positive going transition of the clock pulse not by the level, per se. Now assume that EUCTRDIR = 1 and CP = 1. CPUP therefore becomes 1 to enable the count up lookahead logic.

If the true outputs of all positions to the right of the ith position are 1 then the ith position will change state. More specifically if c_i denotes the clock input to the flip-flop of position i, then the equation

$$\overline{C_i} = \text{CPUP} \cdot \text{EUL}_{i+1} \cdot \text{EUL}_{i+2} \cdot \cdot \cdot \text{EUL}_7$$

is satisfied. EUL_i is the true output of the ith position. C_i drops to logical 0 until CPUP is returned to logical 1. At that time C_i

makes a 0 to 1 transition changing the state of the flip-flop. The inputs to the count-up lookahead logic may change due to a change of the flip-flop from which they are derived, but by this time CPUP = 0 and the lookahead logic is disabled.

The operation of the count down sequence is analogous to the count up sequence. In the previous paragraph merely replace CPUP by CPDWN and all occurrences of EUL by $\overline{EUL}$. The "C" inputs to the EUL Counter are defined as follows:

C_n = "C" input to flip-flop in position n.

Note: If BYONE = 1, then counter increments/decrements by 1, otherwise by two.

$C_7 = CP \cdot BY\emptyset NE$

$C_6 = EUL_7 \cdot (CP \cdot UP) \vee \overline{BY\emptyset NE} \cdot CP \vee \overline{EUL_7} \cdot (CP \cdot DWN)$

$C_5 = EUL_6 \cdot EUL_7 \cdot (CP \cdot UP) \vee \overline{EUL_6} \cdot \overline{EUL_7}\ (CP \cdot DWN)$

$C_4 = EUL_5\ EUL_6\ EUL_7 \cdot (CP \cdot UP) \vee \overline{EUL_5}\ \overline{EUL_6}\ \overline{EUL_7} \cdot (CP \cdot DWN)$

$C_3 = EUL_4 \cdot EUL_5 \cdot EUL_6 \cdot EUL_7 \cdot (CP \cdot UP) \vee \overline{EUL_4} \cdot \overline{EUL_5}$
$\overline{EUL_6} \cdot \overline{EUL_7}\ (CP \cdot DWN)$

$C_2 = EUL_3 \cdot EUL_4 \cdot EUL_5 \cdot EUL_6 \cdot EUL_7\ (CP \cdot UP)$
$\vee\ \overline{EUL_3} \cdot \overline{EUL_4} \cdot \overline{EUL_5} \cdot \overline{EUL_6} \cdot \overline{EUL_7} \cdot (CP \cdot DWN)$

$C_1 = EUL_2 \cdot EUL_3 \cdot EUL_4 \cdot EUL_5 \cdot EUL_6 \cdot EUL_7 \cdot (CP \cdot UP)$
$\vee\ \overline{EUL_2}\ \overline{EUL_3}\ \overline{EUL_4}\ \overline{EUL_5}\ \overline{EUL_6}\ \overline{EUL_7} \cdot (CP \cdot DWN)$

2.12.7 EUL Counter Condition Detection

The EUL Counter-Register drives logic which detects certain conditions necessary for the alignment of the fractional part of floating point operands. The details of this logic is shown on Drawing Nos. 212-04 and 212-05. The condition signals are described in Table 2.12.7.1.

Signal Name	Description	Logic Equation
EUCTØV (Exponent Unit Counter Overflow)	All positions of the counter are 1 and an additional CPUP pulse has arrived	$EUCTØV = (EUL_1 \cdot EUL_2 \cdots EUL_7)_{CPUP}$
EUCTUN (Exponent Unit Counter Underflow)	All positions of the counter are 0 and an additional CPDWN pulse has arrived.	$EUCTUN = \overline{EUL}_1 \cdot \overline{EUL}_2 \cdots \overline{EUL}_7)_{CPDWN}$
EULEZ (EUL Register Equals Zero)	The contents of the EUL Register is zero (in excess 64 notation)	$EULEZ = EUL_1 \cdot \overline{EUL}_2 \cdots \overline{EUL}_7$
DEXPGT13 (Difference in Exponents	The EUL Register-Counter typically contains the difference of the exponent produced by the EU Adder. This signal is true whenever this difference, the contents of EUL, is greater than 13. Note that 13 = 1001101 in excess 64 notation.	$DEXPGT13 = EUL_1 \cdot (EUL_2 \vee EUL_3 \vee EUL_4 \cdot EUL_5 \cdot EUL_6)$
DEXPGEZ (Difference in Exponent Greater than or Equal 2)	The value of the contents of EUL is greater than or equal to +2. Note that Z = 1000010 in excess 64 notation.	$DEXPGEZ = EUL_1 \cdot (EUL_2 \vee EUL_3 \vee EUL_4 \vee EUL_5 \vee EUL_6)$

Table 2.12.7.1--EUL Counter Condition Detection

Signal Name	Description	Logic Equation
DEXPLEM2 (Difference in Exponents Less Than or Equal Minus 2 (-2).)	The value of the contents of EUL is equal to or more negative that minus 2. Note that -2 = 0111110 in excess 64 notation.	DEXPLEM2 = $EUL_1(\overline{EUL_2} \vee \overline{EUL_3} \vee \overline{EUL_4} \vee \overline{EUL_5} \vee \overline{EUL_6} \vee \overline{EUL_7})$
DEXPLTM13 (Difference in Exponent Less Than Minus 13)	The value of the contents of EUL is more negative than minus 13 (-13). Note that -13 = 0110011 in excess 64 notation.	DEXPLTM13 = $\overline{EUL_1} \cdot (\overline{EUL_2} \vee \overline{EUL_3} \vee \overline{EUL_4}\ \overline{EUL_5} \cdot (\overline{EUL_6 \cdot EUL_7}))$
DEXPGTZ (Difference in Exponent than Zero)	The value of the contents of EUL is more positive than zero. This signal is valid even if EUA Overflow occurs.	DEXPGTZ = $EUL_1 \cdot \overline{EULEZ} \cdot \overline{EUAUN} \vee EUAØV$
DEXPLTZ (Difference in Exponent Less Than Zero)	The value of the contents of EUL is more negative than zero. This signal is valid even if EUA Underflow occurs.	DEXPLTZ = $(EUL_1 \cdot \overline{EUAOV}) \vee EUAUN$
DEXPIPR (Difference in Exponent in Positive Range)	The value of the contents of EUL is within the positive range accommodated by the EU without overflow.	DEXPIPR = $EUL_1 \cdot \overline{EULEZ} \cdot \overline{EUAUN} \cdot \overline{DEXPGT13}$
DEXPINR (Difference in Exponent in Negative Range)	The value of the contents of EUL is within the negative range accommodated by the EU without underflow.	DEXPINR = $\overline{EUL_1} \cdot \overline{EUAØV} \cdot \overline{DEXPLTM13}$

```
/*PL/1 DESCRIPTION OF SIGNAL NAMES RELEVANT TO EXPONENT ARITHMETIC
       UNIT (EAU)*/
/*RELEVANT DRAWING NUMBERS: 212-01,-02,-03,-04,-05*/
          DECLARE EUL BIT(7); /* EU LOWER REGISTER/COUNTER*/
          DECLARE EUU BIT(7); /* EU UPPER REGISTER */
          DECLARE EUM BIT(7); /* EU M REGISTER */
          DECLARE EAY BIT(7); /* EXPONENT ADDER 'Y' INPUT */
          DECLARE EUUSEL BIT(7); /*OUTPUT OF SELECTOR GATES
                                   FOR EUU-REGISTER*/
          DECLARE EUMSEL BIT(7); /* OUTPUT OF SELECTOR GATES FOR
                                    EUM-REGISTER */
          DECLARE(DEXPGT13,DEXPLTM13,DEXPGE2,DEXPLEM2) ENTRY
          RETURN (BIT(1));
CP:       /*CLOCK PULSE FOR EAU UP-DOWN COUNTER */
          PROCEDURE(EUCTRDIR,BYONE,EUCTROV,EUCTRUN EULEZ)
          DECLARE (EUCTRDIR,EUCTROV,EUCTRUN,EULEZ,BYONE) BIT(1);
          /* EUCTRDIR = EU COUNTER DIRECTION =1 FOR COUNT UP   */
          /*                                 =0 FOR COUNT DOWN */
          /* BYONE = 1 = COUNT BY ONES                         */
          /*       = 0 = COUNT BY TWOS
          /* EUCTROV = EU COUNTER OVERFLOW (COUNT > 63)
          /* EUCTRUN = EU COUNTER UNDERFLOW (COUNT < -64)
          /* EULEZ = EUL REGISTER CONTENTS = ZERO IN EXCESS-64 */
          /*         NOTATION = '1000000'
             IF(EUCTRDIR ='1'B) & (EUL='1111111'B) THEN
             EUCTROV ='1'B;
             IF(EUCTRDIR ='0'B) & (EUL='0000000'B) THEN
             EUCTRUN = '1'B;
          DECLARE COUNT FIXED BIN;
          COUNT = EUL
          IF BYONE ='1'B & EUCTRDIR = '1'B  THEN COUNT = COUNT +1;
          IF BYONE ='1'B & EUCTRDIR = '0'B  THEN COUNT = COUNT -1;
          IF BYONE ='0'B & EUCTRDIR = '1'B  THEN COUNT = COUNT +2;
          IF BYONE ='0'B & EUCTRDIR = '0'B  THEN COUNT = COUNT -2;
          IF EUL='1000000'B THEN EULEZ ='1'B ELSE EULEZ ='0'B;
          END;
DEXPGE2:  /*DIFFERENCE IN EXPONENT GREATER THAN OR EQUAL 2     */
          PROCEDURE BIT(1);
          DEXPGE2  = SUBSTR(EUL,1,1)       /* DEXP POSITIVE    */
                   &¬(SUBSTR(EUL,2,5)='00000'B;/* ¬= 0 OR 1    */
          RETURN(DEXPGE2);
          END;
DEXPGT13: /*DIFFERENCE IN EXPONENT GREATER THAN 13              */
          PROCEDURE BIT(1);
          DEXPGT13 = SUBSTR(EUL,1,1)       /* DEXP POSITIVE    */
                   &(SUBSTR(EUL,2,1)       /*    > 31          */
                   1 SUBSTR(EUL,3,1)       /*    > 15          */
                   1(SUBSTR(EUL,4,3)='111'B)); /*  = 14 OR 15  */
          RETURN (DEXPGT13);
          END;
DEXPLEM2: /*DIFFERENCE IN EXPONENT LESS THAN OR EQUAL TO -2     */
          PROCEDURE BIT(1);
          DEXPLEM2 = ¬SUBSTR(EUL,1,1)          /*EUL NEGATIVE */
                   &¬(SUBSTR(EUL,2,6)='111111'B;/* ¬= -1       */
          RETURN(DEXPLEM2);
          END;
DEXPLTM13: /*DIFFERENCE IN EXPONENT LESS THAN -13               */
```

```
        PROCEDURE BIT(1);
        DEXPLTM13 =(¬SUBSTR(EUL,1,1)          /*EUL NEGATIVE */
                  &(¬SUBSTR(EUL,2,1)          /* < -32       */
                  | ¬SUBSTR(EUL,3,1))         /* < -16       */
                  | (SUBSTR(EUL,4,2)='00'B    /* = -14,-15,  */
                  &¬ SUBSTR(EUL,6,2)='11'B)); /* OR -16      */
        RETURN(DEXPLTM13);
        END;
ADEUU:  /*EXPONENT OF A-OPERAND TO EUU-REGISTER*/
        PROCEDURE;
        EUU=SUBSTR(V,12,7);
        END;
BDEUM:  /*EXPONENT OF B-OPERAND TO EUM-REGISTER*/
        PROCEDURE;
        EUM=SUBSTR(V,12,7);
        END;
UA:     /*EXPONENT UNIT ADDER. INCLUDES OVERFLOW DETECT.*/
        PROCEDURE (EUM,EUU,ES,EUDIFF,EUAOV,EUAUN);
        DCL (EUM,EUU,ES) BIT(7), EUDIFF BIT(1),
            (EUAOV,EUAUN) BIT(1), EAY BIT(7), C BIT(7);
        /*EUM,EUU  ARE REGISTERS PREVIOUSLY DEFINED */
        /*ES IS EXPONENT ADDER SUM OUTPUT */
        /*EUDIFF IS '1'B IF DIFFERENCE OF EXPONENTS IS TO BE
          FORMED; IS '0'B OTHERWISE */
        /*EUAOV = EXPONENT UNIT ADDER OVERFLOW */
        /*EUAUN = EXPONENT UNIT ADDER UNDERFLOW */
        /*C IS A TEMPORARY STRING USED TO STORE CARRIES */
        /*IMPLEMENTED WITH T.I. I.C. NOS: SN7482N AND SN7483N */
        IF(EUDIFF = '1'B) THEN  EAY= ¬EUM  ELSE EAY = EUM;
        SUBSTR(C,7,1)=EUDIFF;
        DO I=6 TO 1; II= I+1; /* CARRY RIPPLE */
        SUBSTR(C,I,1)=SUBSTR(EAY,II,1)&SUBSTR(EUU,II,1)|
                      SUBSTR(EAY,II,1)&SUBSTR(C,II,1)  |
                      SUBSTR(EUU,II,1)&SUBSTR(C,II,1)  ;
        EUAOV=  SUBSTR(EUU,1,1) & SUBSTR(EAY,1,1) &
                SUBSTR(C,1,1);
        EUAUN= ¬SUBSTR(EUU,1,1) &¬SUBSTR(EAY,1,1) &
               ¬SUBSTR(C,1,1);
        /*NEED TO COMPLEMENT CARRY INTO POSITION 1 IN ORDER
          TO PRODUCE EXCESS 64 RESULT, THUS: */
        SUBSTR(C,1,1)=¬SUBSTR(C,1,1);
        ES=(EAY&¬EUU|¬EAY&EUU)&¬C |
           (EAY&EUU |¬EAY&¬EUU)&C;
        RETURN;
        END;
ULDEUU: /* EUL-REGISTER DIRECT TO EUU-REGISTER*/
        PROCEDURE;
        EUU=EUL;
        END;
UXDEUM: /*EUX REGISTER DIRECT TO EUM REGISTER */
        PROCEDURE;
        EUM=EUX;
        END;
XDEUX:  /*EXPONENT OF X (ARGUMENT OF POLY) DIRECT EUX REGISTER) */
        PROCEDURE;
        EUX=SUBSTR(V,12,7);
        END;
```

```
LDEUL:      /*LOAD EUL REGISTER-COUNTER */
            PROCEDURE;
            EUL=ES;
            END;
LDEUM:      /*LOAD EUM REGISTER */
            PROCEDURE;
            EUM=EUMSEL;
            END;
LDEUU:      /* LOAD EUUSEL INTO EUU-REGISTER */
            PROCEDURE;
            EUU=EUUSEL;
            END;
```

2.13 Control

2.13.1 General

The control of the arithmetic units, implemented in a pseudo-asynchronous fashion, is based on the concept of control steps. A control step consists of two stages: the sequence stage followed by the task stage. The distinction between the sequence and task stage is somewhat analogous to the distinction made between the fetch and execute cycles of a central processor control unit. In further defining these stages consider first the task stage. Here action, conditional upon status conditions, is performed on the processing hardware: i.e. flip-flops are set, gates operated, counters incremented, so as to move data. The sequence stages are interleaved with the task stages. Within the sequence stage the decision is made as to which task stages to next initiate.

2.13.2 Task Stage Logic (Control Point)

This section describes the logic associated with the task stage of a control step. This group of logic is also referred to as a control point.

First consider the block diagram of typical task stage logic as shown in Figure 2.13.2.1. The $\overline{\text{Set-Go}}$ ($\overline{\text{S-G}}$) line is given by the adjacent sequence stage logic. When this line drops to "0", the memory element is set and the task logic is said to be primed. When this line returns to "1", and if Enable is "1" then the DO signal becomes "1" allowing the task signals to be operated subject to the appropriate external conditions. The return of the $\overline{\text{S-G}}$ line to "1" is said to initiate the task stage logic.

The signal $\overline{\text{DO}}$ from the memory element box is one input to the conditional task logic. The other inputs are external conditions appropriate to the task stage. Typical examples of these conditions are outputs of the instruction variant decoder, the contents of a register equal zero, or the output of status flip-flops.

The DO signal also activates the reply generation logic. In most cases the reply is generated by an internal delay element which, after a selectable duration, resets the memory element thus turning off the task element. The delay element provides a timing model of the actual task. In some cases an external reply is available. In many cases all tasks of a given task step will require about the same interval of time. In this case a single timing model will suffice. If this is not the case, then multiple models are provided and activated conditional upon which tasks have been enabled by external conditions. The outputs of the timing model must be logically combined to produce a "1" to "0" drop on the $\overline{\text{Reset-Advance}}$ ($\overline{\text{R-A}}$) line when all tasks have been completed. The $\overline{\text{R-A}}$ signal may therefore be a function of DO, external conditions, and external replies. The $\overline{\text{R-A}}$ line dropping to "0" resets the memory element which turns off the task signals. The R-A signal drives the next sequence stage logic which in turn primes and initiates the next task stage.

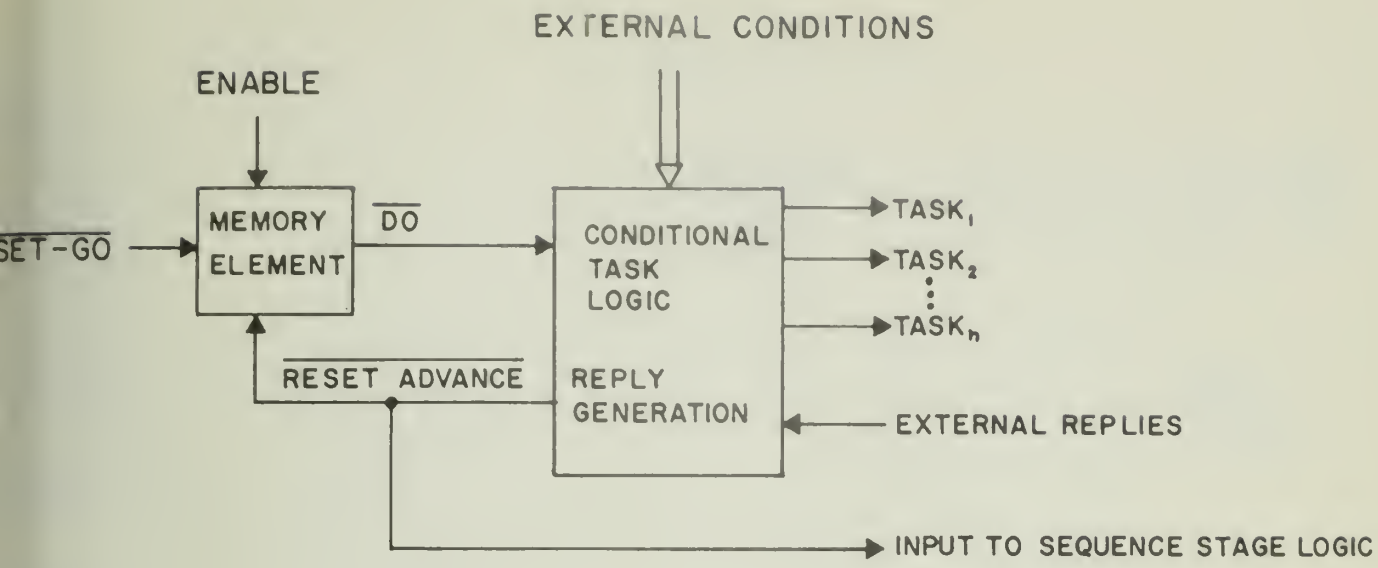


Figure 2.13.2.1 Block Diagram of Task Stage Logic (Control Point)

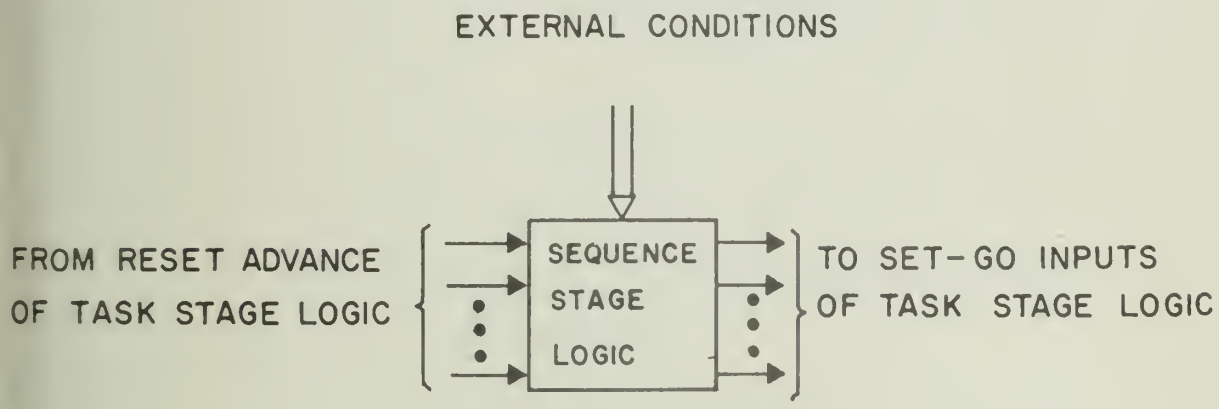


Figure 2.13.2.2 Block Diagram of Sequence Stage Logic

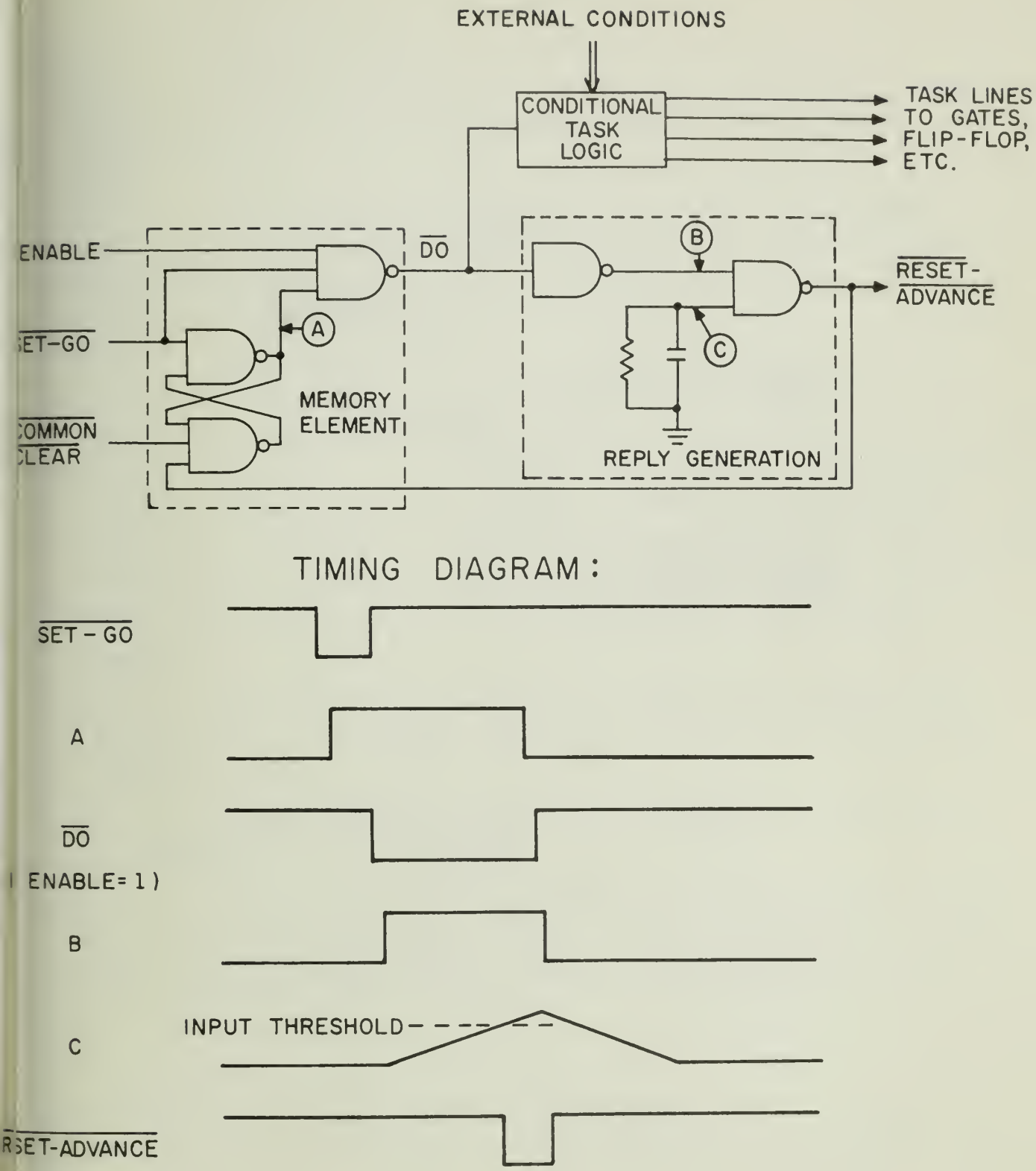


Figure 2.13.2.3 - Most Elementary Task Stage Configuration

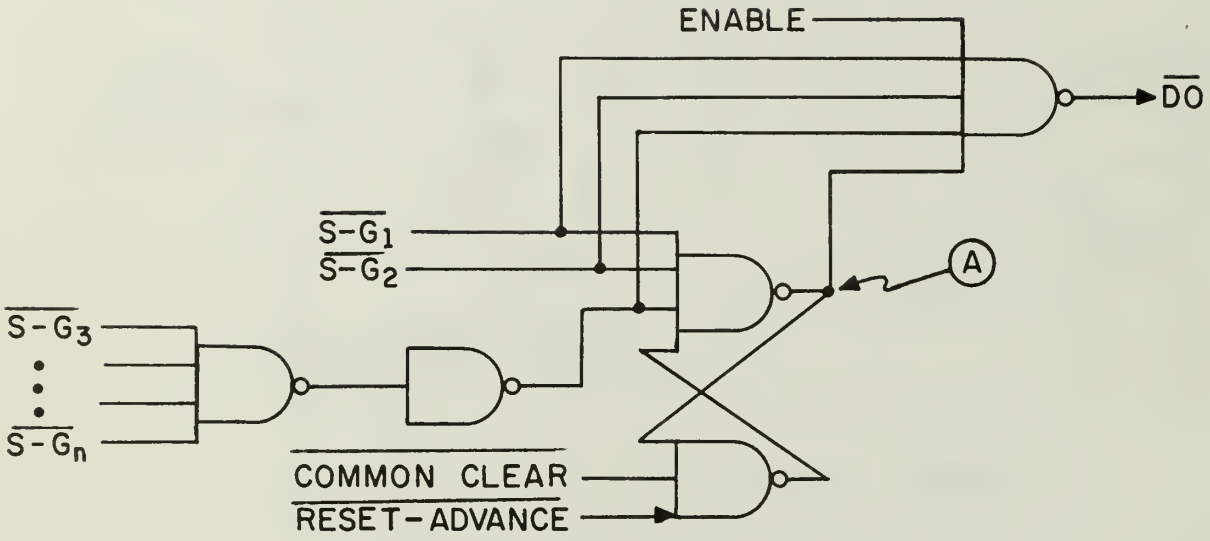


Figure 2.13.2.4 - Multiple Set-Go Inputs to Memory Element

2.13.3 Sequence Stage Logic

The sequence stages are interleaved between task stages. Within the sequence stage the decision is made as to which task stage(s) to next initiate.

The most elementary example of sequence stage logic is merely a wire from the Reset-Advance output of one task stage connected to the Set-Go input of the next task stage. In this case, the two tasks unconditionally follow one another.

Figure 2.13.3.1 illustrates the hardware and operation of a two way branch. The conditions, α and β, determine where the R-A pulse from the previous task is directed. Note that these must be set prior to the arrival of the pulse. It is generally not advisable that α or β be determined by actions in the task stage which generates the R-A pulse which they steer. Furthermore note that for a conditional branch, at least one condition must be true. The fourth R-A pulse in Figure 2.13.3.1 illustrates this case; since $\alpha = \beta = 0$, the pulse is lost and the control sequence hangs-up.

Notice that the α or β signals in Figure 2.13.3.1 cannot be used as wait signals, i.e. to delay continuation of control until an asynchronous condition is true.

Figure 2.13.3.2 illustrates a way in which a wait condition may be implemented.

In some cases it may be required that an external control signal, for example from another unit, initiates successive task stages. In general the arrival time of this signal is unknown. The example given in Figure

---

*The transition of $\overline{\text{R-A}}$ from 1 to 0 and back to 1.

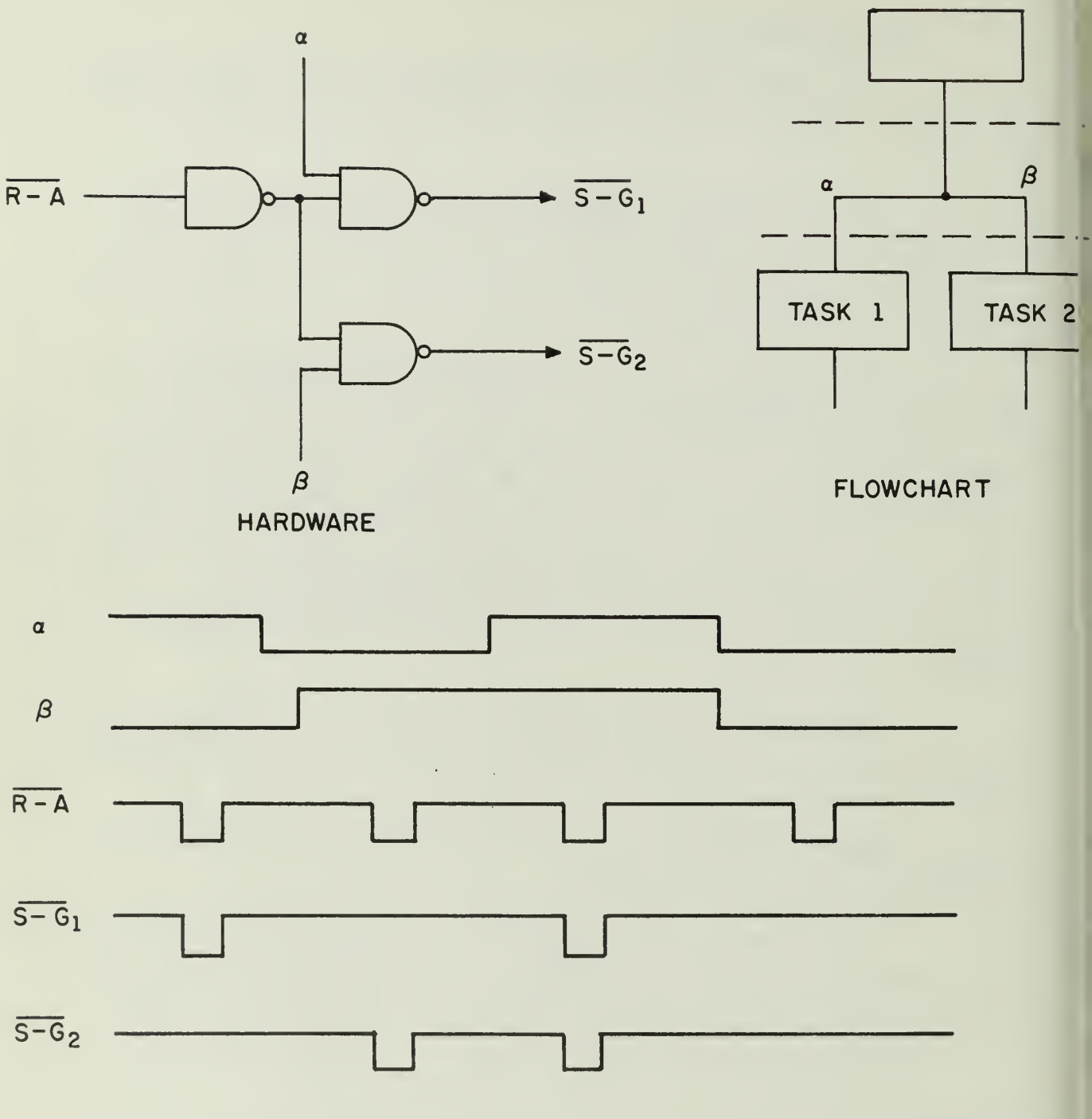


Figure 2.13.3.1 An Example of Sequence Stage Logic

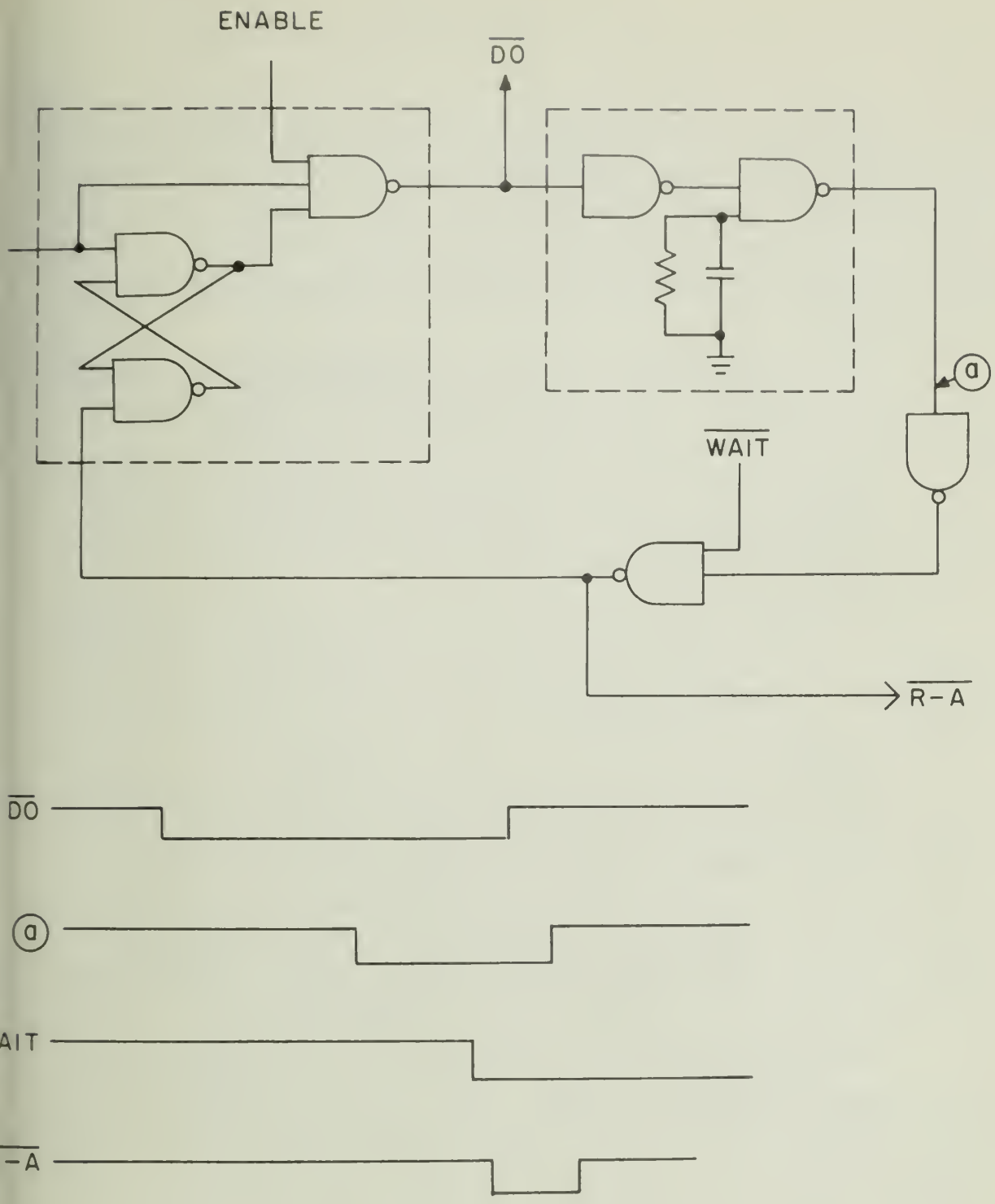


Figure 2.13.3.2 - Implementation of WAIT Condition

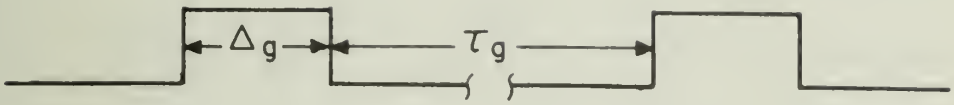
2.13.3.3 illustrates how this may be done using wait logic as in Figure 2.13.3.2 and the ENABLE input. There are no internal reply generation logic; the reply is generated when g, the gate control signal, drops. This logic is actually nothing more than wait logic. The control point reply is generated one delay after g becomes 1 but the R-A signal waits for g to again be 0.

The wait logic may also be used to interlock two or more parallel, independent control chains. The design task here is to make the S-G signal to the next task stage wait until all of the parallel tasks are complete. Figure 2.13.3.4 illustrates an example of interlocked control chains. The Reset to the last task stage of each chain is delayed until the reply from all of the task stages is received.

The scheme in Figure 2.13.3.4 assumes that when one control chain is activated, then so is the other, i.e. that eventually both replies will be generated. Now consider two parallel chains, one which is always activated and another which is activated if same condition α is true.

If $\alpha = 1$, then both chains are activated and the reply from the last task stage of CHAIN A is complemented by the exclusive-OR gate, i.e. the exclusive-OR performs the same function as the inverter in CHAIN B. If $\alpha = 0$, then CHAIN A will not be activated and therefore the reply signal, a, will remain at 1. If this signal were coupled to an input of the $NAND_1$ gate merely by an inverter, this input would remain at 0 and the R-A signal would not be generated even when the reply from CHAIN B arrived. When $\alpha = 0$ and $a = 1$, the exclusive-OR circuit produces an output at C of 1, thus in a sense produces a pseudo reply to $NAND_1$. The R-A pulse will then be produced when the reply from CHAIN B arrives. Note, however, that α must remain set throughout the execution of CHAIN B. An example of this approach is shown in Figure 2.13.3.5.

g = external signal of following form



τ_g is unknown, although greater than several propagation delays.

Δ_g is greater than time required to perform task designated by g

Example: Load X-Register on first g pulse;
Load Y-Register on second g pulse.

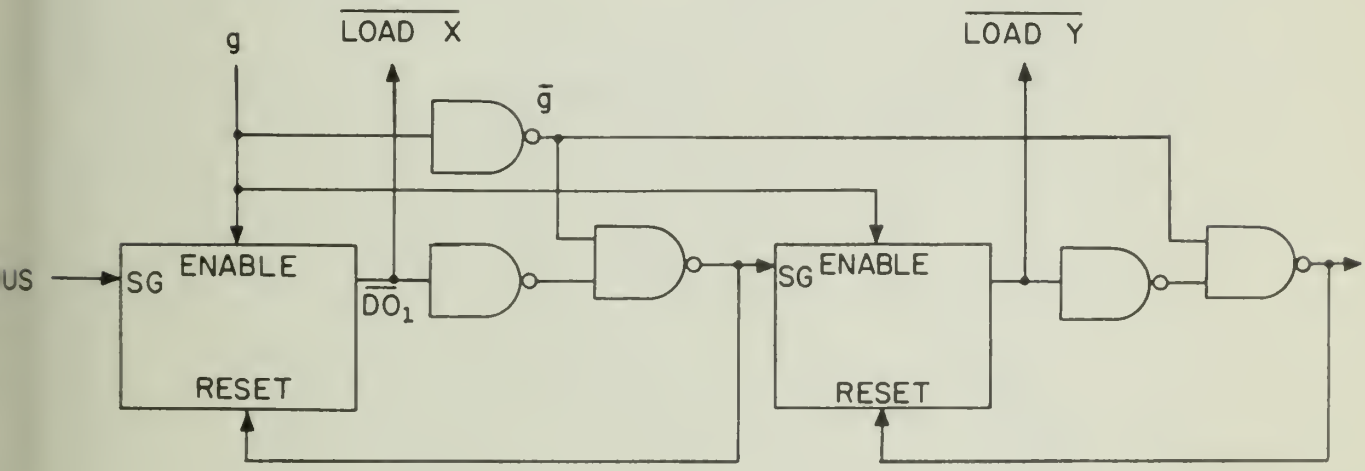


Figure 2.13.3.3 - Control Logic Sequenced by External Signal

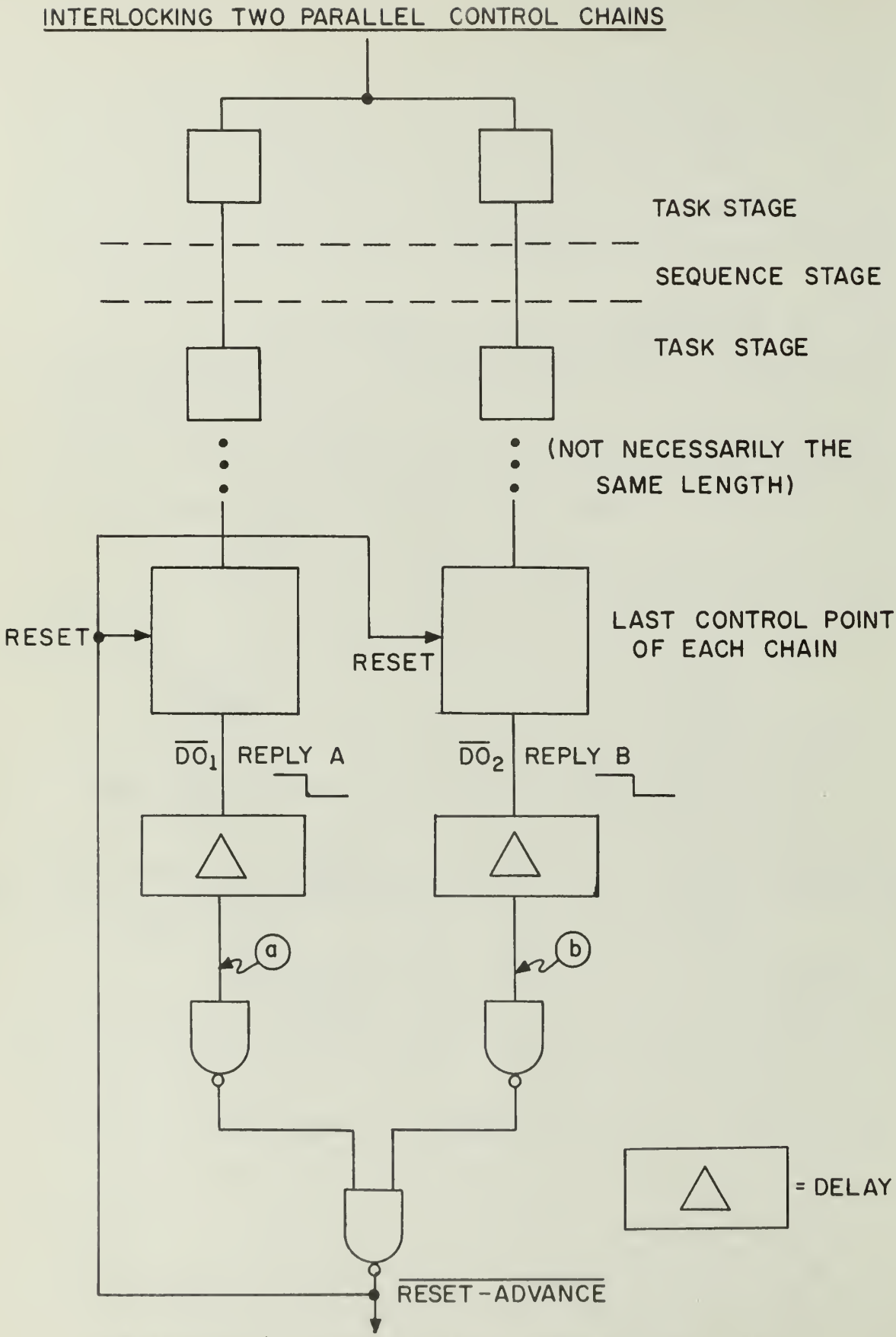


Figure 2.13.3.4 - Interlocking Two Parallel Control Chains

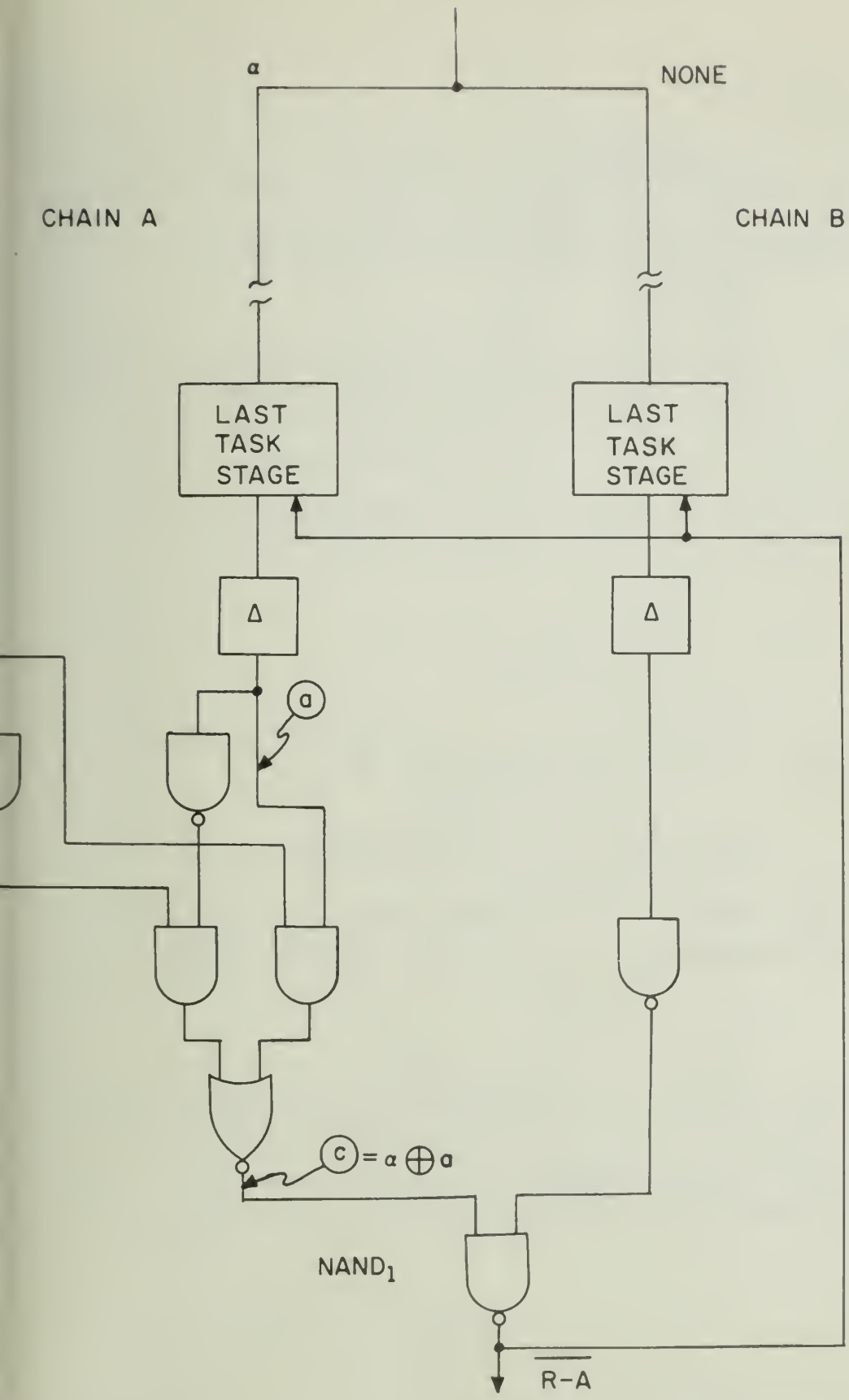


Figure 2.13.3.5 - Interlocking Two Parallel Control Chains - One Unconditional, One Conditional

2.13.4 Reply Generation

In most cases reply signals indicating that a task is complete are generated by delaying the DO signal. The delay is accomplished by the logic shown in Figure 2.13.4.1. The following is design information on this configuration, taken from a memo of October 22, 1968, by J. L. Divilbiss.

Figure 2.13.4.2 is a simplified representation of NAND gate II in Figure 2.13.4.1 and the equivalent circuit used in this analysis. Now assume that the input (a) is at zero volts for several time constants (analysis the same even if V_{IN} = +.4) and furthermore assume an input threshold of 1.4 volts, the midpoint of T.I.'s uncertainty range of .8 to 2 v. Consider case 1 for which C = 100 pF, R = 8K. Therefore,

$$V_{TH} = 3.33v., \quad R_{TH} = 2.67K, \text{ and}$$

$$V_C = V_{TH} (1 - e^{-t/(R_{TH}C)})$$

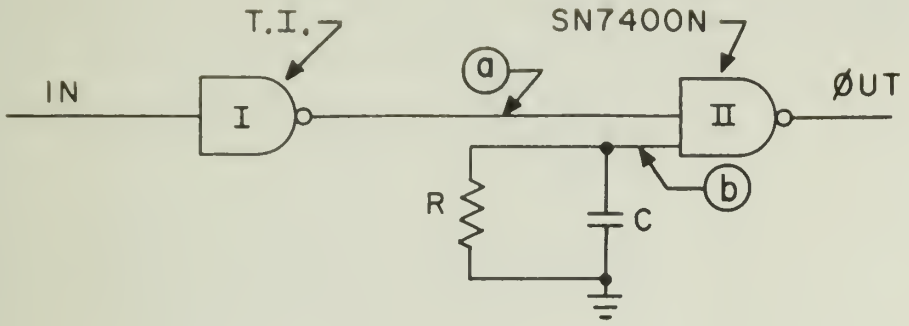
Delay ends when V_C = threshold = 1.4, thus $1.4 = \frac{10}{3} (1 - e^{-t/(R_{TH}C)})$

$$t = .545 \times 2.67 \times 10^3 \times 10^{-10} = 145 \text{ ns.}$$

For case 2, let C = 100 pF, R = 16K and thus V_{TH} = 4 v. and R_{TH} = 3.2K. By a similar derivation

$$t = 138 \text{ ns.}$$

Note that from case 1 to case 2, R was increased from 8K to 16K but that the delay time increased only slightly. The recovery time (the time for C to discharge) has, however, doubled. The recover time is determined solely by R and C. About four RC time constants will discharge C



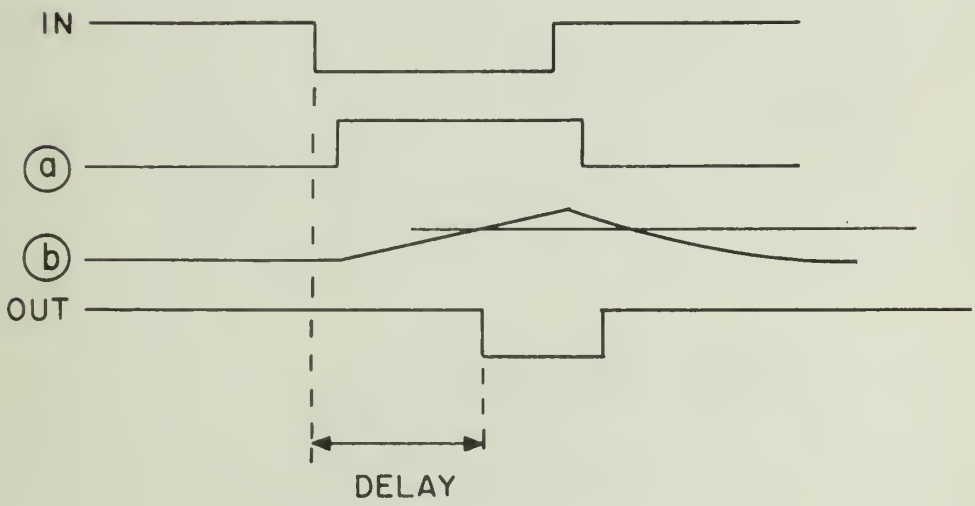
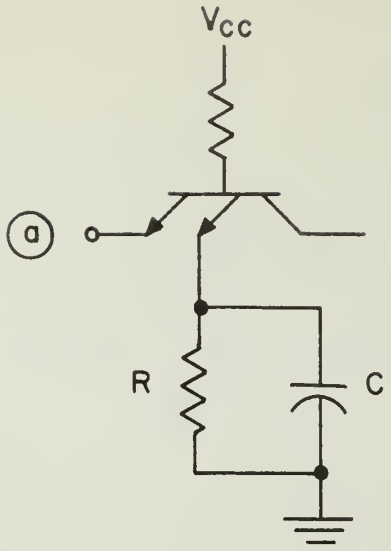
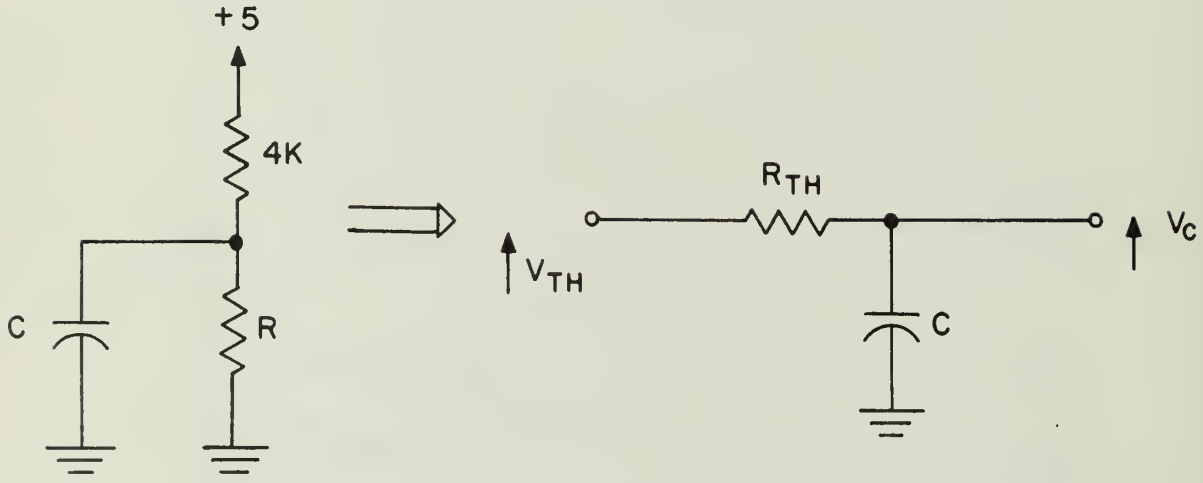


Figure 2.13.4/1 - Delay Element for Reply Generation



EQUIVALENT CIRCUIT



WHERE $V_{TH} = \frac{5R}{R+4}$

$R_{TH} = \frac{4R}{R+4}$

$V_C = V_{TH}(1 - e^{\frac{-t}{RC}})$

Figure 2.13.4/2 - Equivalent Circuit for Analysis

sufficiently to avoid interaction between pulses. This fact argues strongly that R should be made as small as possible consistent with waveform distortion. Empirical tests have indicated a design center value of 8.2K with upper and lower limits of 12K and 6.2K.

With R = 8.2K, the following holds:

Delay - 1.5 ns/pF
Variation for $\pm$ .5v in V_{cc} - $\pm$ 15%
Variation between IC packages - $\pm$ 15%
(Sample of seven)

Variation due to change of R in the range 6.2K to 12K is $\pm$ 8%.

For the circuit shown in Figure 2.13.4.2, approximately 12 delay times must elapse between input pulses to avoid interaction. If this constraint hinders performance it may be greatly reduced by the addition of a diode as shown in Figure 2.13.4.3. The USD25 is an HP hot carrier device with a low forward drop. With the addition of the diode, it is necessary to allow 3 delay times between input pulses.

Figure 2.13.4.4 illustrates a variation of Figure 2.13.4.1. This configuration delays the "0" to "1" transition rather than the "1" to "0" transition. Note, however, that if the diode is used to enhance recovery, this second configuration is not recommended since it severely loads the IN signal.

Having looked at these electronic delay elements in some detail, we now consider their use in reply generation. In many cases only one timing model and thus only one delay element is necessary for a given task stage. In other cases one of several timing models is selected depending upon which tasks at a particular task stage are selected. Figure 2.13.4.5 illustrates a scheme which will meet this requirement provided only one timing model is enabled at a time. If more than one is enabled, the shortest delay will predominate. This is generally not the performance wanted.

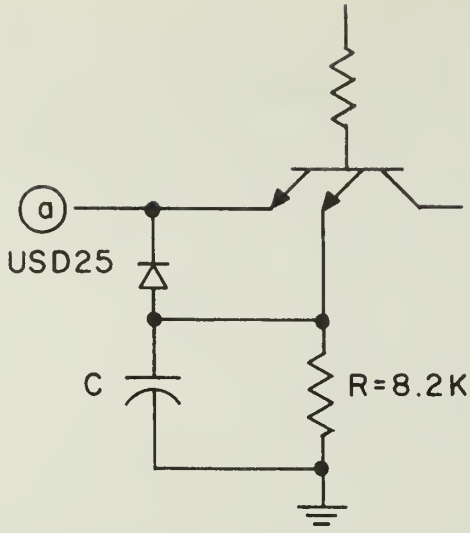


Figure 2.13.4.3 Delay Circuit with Diode to Enhance Recovery

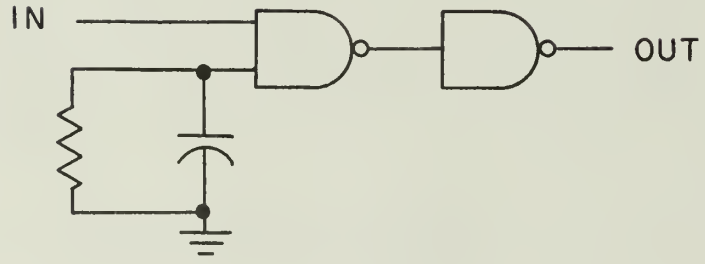
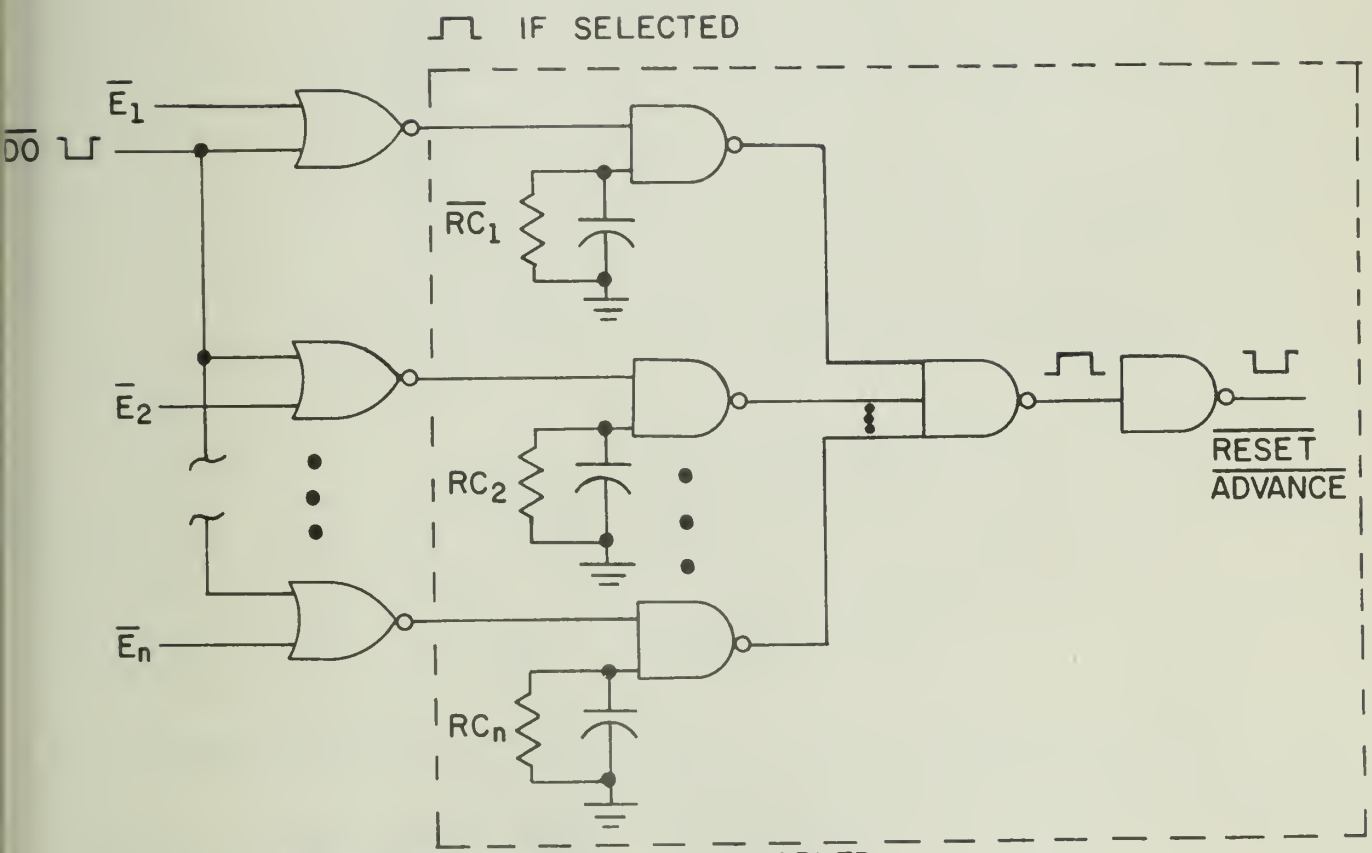


Figure 2.13.4/4 - Delay Element to Delay "0" to "1" Transition



IF $E_i = 1$, THEN TIMING MODEL i IS ENABLED

ASSUMPTION: ONLY ONE $E_i = 1$

NOTE: LOGIC WITHIN [ ] MAY BE REPLACED BY AN AND-NOR COMPLEX SUCH AS AN SN7451N.

Figure 2.13.4/5 - Selectable Timing Models

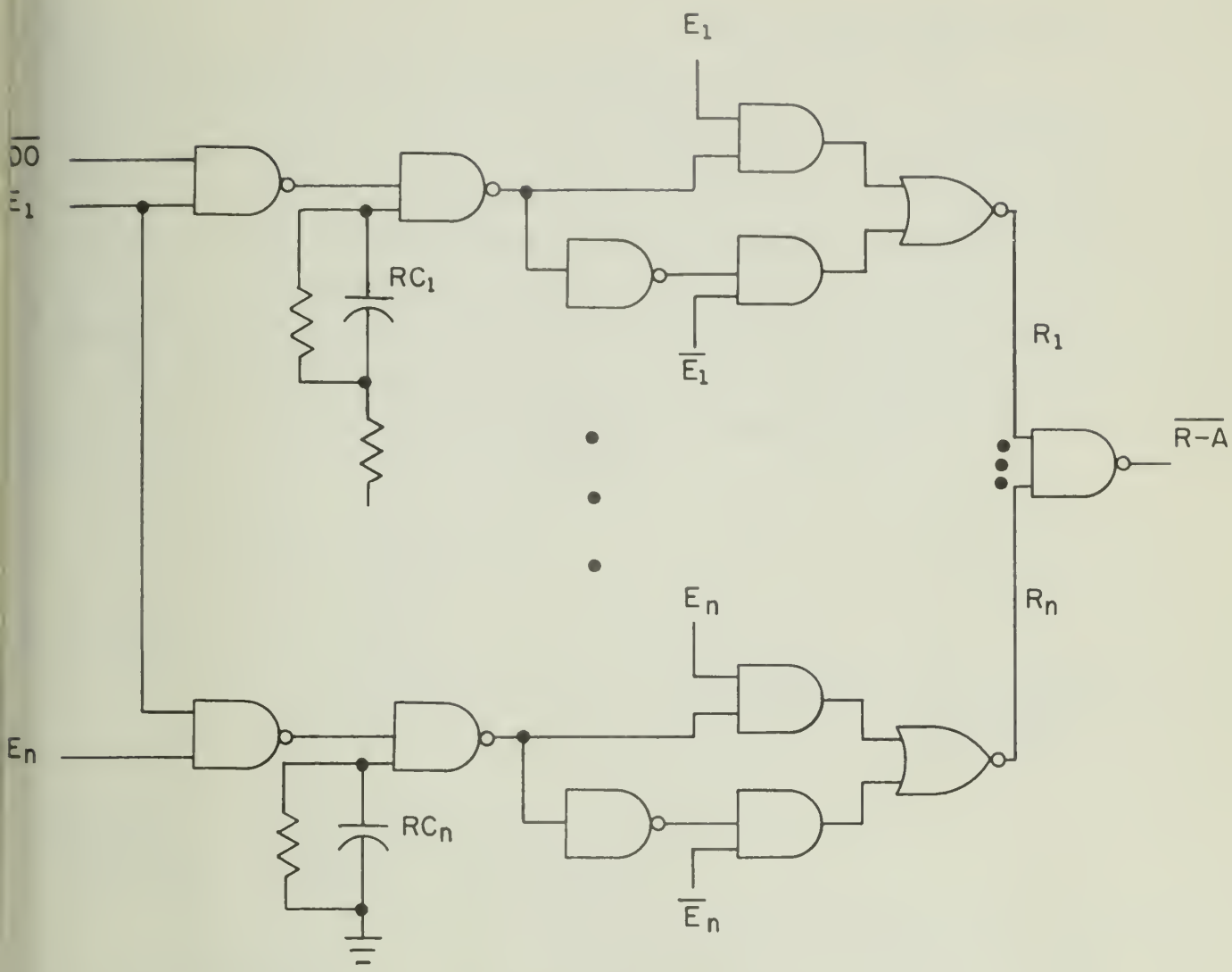
When more than one timing model is activated, the reset-advance pulse should not be generated until the replies from all timing models are received. Figure 2.13.4.6 illustrates a scheme which will meet this requirement. The signal $\overline{R\text{-}A}$ drops to "0" when all replies, R_1 through R_n are "1". For all timing models which are <u>not</u> enabled, the reply is immediately set to "1". For all timing models which are enabled, the reply R will go to "1" after the delay time injected by the timing model.

Another scheme for handling multiple timing models has been suggested by L. Goyal. Let R_i be the output of the i^{th} delay element. Once DO = "1", R_i = 1 after time Δ_i. Now order the numbering of the delay elements such that a higher number denotes a longer delay, i.e. for delay element i with output R_i and delay time Δ_i, if $j > k$ then $\Delta_j > \Delta_k$. Let E_i be the task enable signal associated with Δ_i and R_i. Let the maximum subscript (number of delay elements) be denoted n.

Then

$$\begin{aligned} \text{Reset-Advance} = {} & R_n \vee \overline{E}_n \; R_{n-1} \\ & \vee \overline{E}_n \; \overline{E}_{n-1} \; R_{n-2} \\ & \vee \ldots \\ & \vee \overline{E}_n \; \overline{E}_{n-1} \ldots \overline{E}_2 \; R_1 \end{aligned}$$

We have derived priority logic; the timing model with the longest delay of those selected is the one used to produce Reset-Advance.



IF $E_i = 1$, THEN TIMING MODEL i IS ENABLED.

Figure 2.13.4.6 - Selectable Timing Model: Reset-Advance Waits Until Longest Reply is Received

2.13.5 Control Logic Layout

The control logic is implemented with Texas Instruments 7400N Series IC logic. Two etched boards have been designated and are illustrated in Figures 2.13.5.1 and 2.13.5.2. These two boards and the auxiliary timing model board will constitute the bulk of the task stage logic. Sequence stage logic, task condition logic, and any reply generation logic other than a single delay element will be implemented on wirewrap connector boards. This board provides facilities for 24, dual inline packages (14 or 16 pin) and includes 44 card edge pins which mate with connectors on the mainframe.

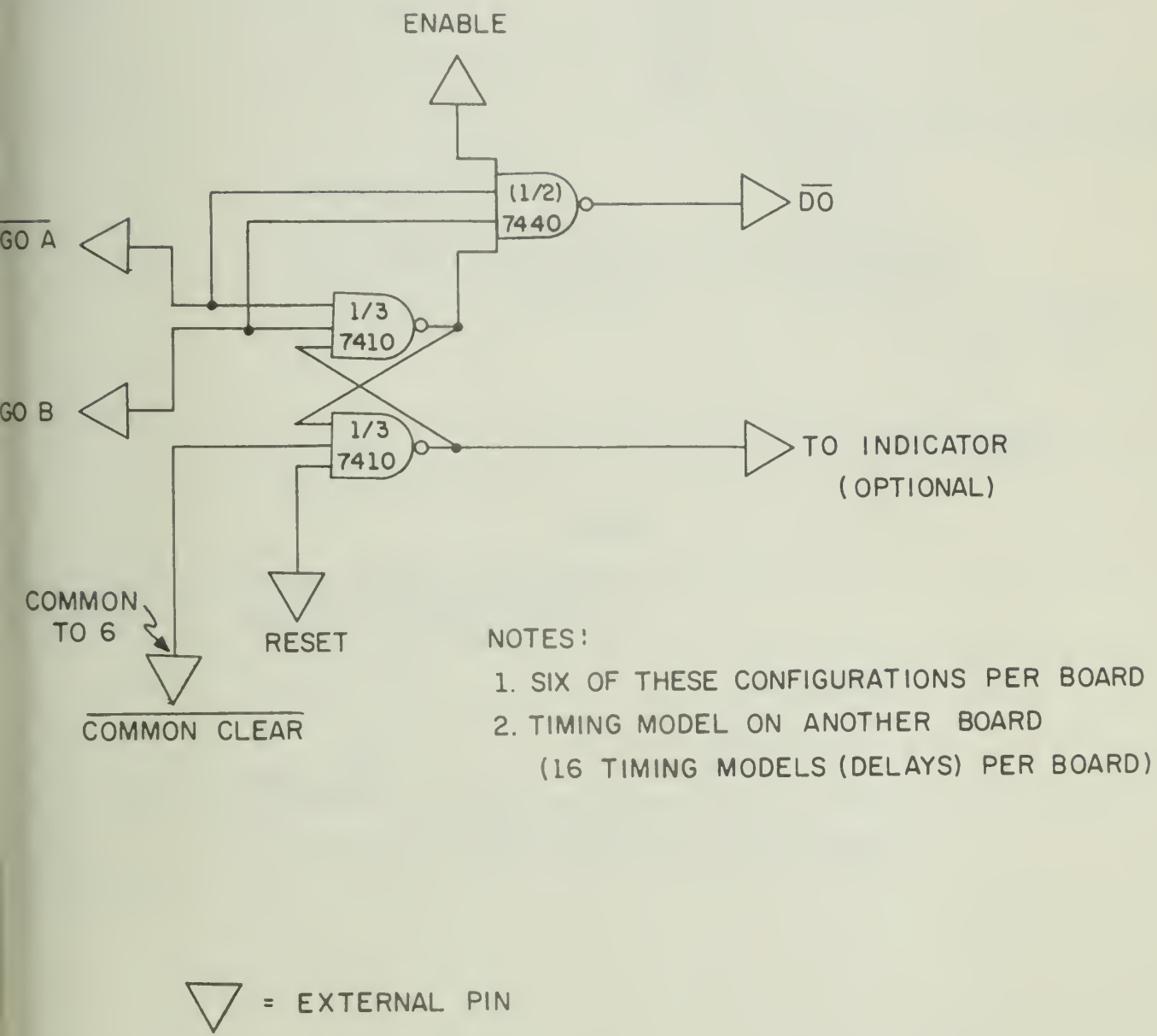
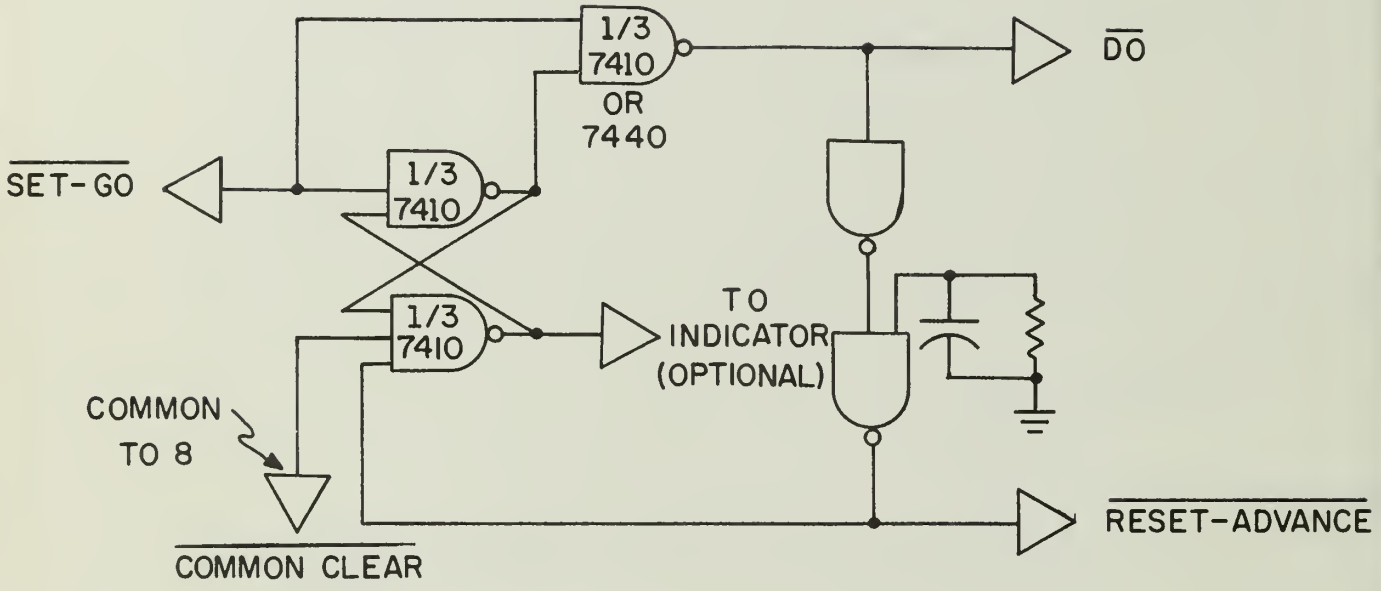


Figure 2.13.5.1 - Control Point Card, Version A



NOTES:

1. EIGHT OF THESE CONFIGURATIONS PER
2. TIMING MODEL INCLUDED ON BOARD

▽ = EXTERNAL PIN

Figure 2.13.5.2 - Control Point Card, Version B

m AEC-427
(6/68)
ECM 3201

U.S. ATOMIC ENERGY COMMISSION

UNIVERSITY–TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIF'C AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

AEC REPORT NO.	2. TITLE
[illegible]-1018-1194	Illiac III Computer System Manual: Arithmetic Units, Vol. 1

TYPE OF DOCUMENT (Check one):

- [x] a. Scientific and technical report
- [ ] b. Conference paper not to be published in a journal:
  - Title of conference _____
  - Date of conference _____
  - Exact location of conference _____
  - Sponsoring organization _____
- [ ] c. Other (Specify) _____

RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- [x] a. AEC's normal announcement and distribution procedures may be followed.
- [ ] b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
- [ ] c. Make no announcement or distrubution.

REASON FOR RECOMMENDED RESTRICTIONS:

SUBMITTED BY: NAME AND POSITION (Please print or type)

Daniel E. Atkins

Organization

Department of Computer Science
University of Illinois
Urbana, Illinois 61820

Signature	Date
D.E. Atkins III	

FOR AEC USE ONLY

AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

PATENT CLEARANCE:

- [ ] a. AEC patent clearance has been granted by responsible AEC patent group.
- [ ] b. Report has been sent to responsible AEC patent group for clearance.
- [ ] c. Patent clearance not required.